Vertex Deletion on Chordal Graphs and Their Subclasses

A Thesis

submitted to
Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements for the
BS-MS Dual Degree Programme

by

Anirudh Raghava Rachuri

Indian Institute of Science Education and Research Pune
Dr. Homi Bhabha Road,
Pashan, Pune 411008, INDIA.

April, 2023

Supervisor: Dr. Soumen Maity Co-Supervisor: Prof. Saket Saurabh

Certificate

This is to certify that this dissertation entitled 'Vertex Deletion on Chordal Graphs and Their Subclasses' towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Anirudh Raghava Rachuri at Indian Institute of Science Education and Research, Pune under the supervision of Dr. Soumen Maity, Associate Professor, Department of Mathematics, Indian Institute of Science Education and Research, Pune and Prof. Saket Saurabh, Professor, Department of Theoretical Computer Science, Institute of Mathematical Sciences, Chennai, during the academic year 2022-2023.

Dr. Soumen Maity

Prof. Saket Saurabh

Committee:

Dr. Soumen Maity

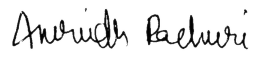
Prof. Saket Saurabh

Dr. Vivek Mohan Mallick

To my family and friends, this would not have been possible without you

Declaration

I hereby declare that the matter embodied in the report entitled 'Vertex Deletion on Chordal Graphs and Their Subclasses', are the results of the work carried out by me at the Department of Mathematics, Indian Institute of Science Education and Research, Pune, Indian Institute of Science Education and Research, Pune, under the supervision of Dr. Soumen Maity and Prof. Saket Saurabh and the same has not been submitted elsewhere for any other degree.



Anirudh Raghava Rachuri

Acknowledgments

I would like to express my sincere gratitude to all those who have supported me throughout my academic journey and the completion of my thesis.

I am deeply grateful to my parents for their unwavering support, encouragement, and belief in me. Their guidance and sacrifices have been instrumental in shaping me into the person I am today. I would also like to thank my sister for her constant support and motivation.

I am grateful to my project supervisors, Prof. Saket Saurabh and Dr. Soumen Maity for their guidance, constructive feedback, and patience throughout the duration of my research project. Their work ethic, insights and expertise have been invaluable and contributed significantly to the completion of this thesis. I would also like to express my appreciation to Ajinkya Gaikwad, whose advice and insights were immensely helpful throughout the course of this project. I am thankful to the DST-INSPIRE program for its financial support via the Scholarship for Higher Education (SHE).

I extend my gratitude to my colleagues Mihir Neve, Hrishikesh V and Ajaykrishnan ES for always listening to my ideas and thoughts and offering their valuable inputs. I will dearly miss all the fruitful (as well as the meaningless!) discussions we had. I would also like to thank my friends, S Rohan, Shaswat Nair, Chahana Nagesh and others, for always being there for me. I am incredibly fortunate to have met an amazing set of friends, and I will miss all of the fun we had in these 5 years.

Once again, thank you to all those who have supported me in my academic pursuits. I am deeply grateful for your contributions and encouragement.

Abstract

Numerous computational problems on graphs remain computationally intractable and are termed NP-complete problems. In this thesis, we study one of the ways to tackle this issue: we restrict the input graphs by specifying certain properties and exploit these to build efficient algorithms on these classes. We focus on the vital class of chordal graphs, which have rich structural properties with varied algorithmic applications.

A major class of graph problems are termed vertex deletion problems, which involve eliminating a certain set of vertices so that the resultant graph satisfies some properties. We study vertex deletion problems on the chordal graph class, as well as their subclasses. We first review the relevant properties of these graph classes and study the status of vertex deletion problems on these graphs. We then move to design efficient algorithms for some unresolved problems.

Contents

Chapter 1

Introduction

A *graph* G is a pair of sets (V, E) such that $E \subseteq V \times V$, where V is the set of vertices and E is the set of edges. Each element of E joins two elements of V, and is called an edge of G. Graph theory is extensively studied from a structural and algorithmic point of view, with the two approaches often proving helpful to each other.

Numerous computational problems have been proposed on graphs, which have vital practical implications. Graphs can be used to model relationships between objects, with the vertices being the objects and the edges being the relationship between them. For instance, the vertex cover problem, which asks for a subset S of vertices such that every edge has an endpoint in S, has various practical implications. As a toy example, consider a communication network modelled using graphs. If the vertices of the graph are nodes of the communication network, protecting the nodes in the vertex cover ensures that the communication network does not break down.

Various diverse algorithmic problems have been proposed on graphs; many are still unresolved or have no known fast algorithms. For such computational problems on graphs, numerous ways exist to work around the issue of intractability. Here, we shall study one of the popular ways to tackle this problem: we restrict ourselves to specific graph classes, such that we can exploit their properties and design more efficient algorithms for them. The advantages of this technique are that there is a clear pathway to attacking this problem by exploiting the properties of the restricted input. Many algorithms also provide structural insight into these inputs. However, a drawback is that we only solve the problem for a subset

of all inputs, and the algorithm does not give a solution for any arbitrary input. Nevertheless, this approach remains vital in understanding the tractability of various graph problems. In particular, if we can prove that the problem remains hard on a class of inputs, we can easily deduce that the problem is hard for any general input.

In this regard, one of the most popular graph classes under study is the class of chordal graphs, which are graphs that contain no cycle of length greater than 3 as an induced subgraph. Chordal graphs have rich structural properties, many of which have been used to build algorithms ([1, Chapter 4]). Chordal graphs are also useful in understanding the structural properties of other graph classes due to their close relationship with the concept of treewidth ([2]). Owing to these important properties, chordal graphs generate a lot of interest and are extensively studied.

Several subclasses of chordal graphs have also been well-studied, as they not only share the properties of chordal graphs, but have additional structure imposed on them, making them very well-behaved. One of the most important subclasses of chordal graphs are the split graphs. Split graphs are chordal graphs with the additional requirement that the vertex set be partitioned into a complete graph and an independent set. This property makes split graphs extremely important from an algorithmic point of view, as various hard problems on chordal graphs have efficient solutions on split graphs; for instance, see [3].

To close out the complexity gap between chordal and split graphs, a generalisation of split graphs was introduced in [4], called well-partitioned chordal graphs. These graphs are a subclass of chordal graphs and a superclass of split graphs, maintaining and extending the properties of both these subclasses. They form an extremely important tool in narrowing down the complexity gap of problems on chordal graphs and their subclasses (for instance, see[4]).

We examine a subset of graph theory problems known as vertex deletion problems, which have a wide range of applications in areas including computer science, operations research, and social network analysis. These are problems where we must identify a subset of the vertex set whose elimination guarantees that the remaining graph meets a particular requirement. In this thesis, the vertex deletion problems studied are such that we demand the output be a certain graph class (such as split graphs, for instance). Stated simply, we are transforming an input graph into a graph that satisfies some criteria by eliminating the fewest amount of vertices. Although vertex deletion problems are well-studied on chordal graphs and their

subclasses ([5]), they have not been examined on the relatively new well-partitioned chordal graph class.

We will start with a basic exposition on chordal graphs, as well as some of their subclasses. We will then introduce vertex deletion problems, and look at some examples of these problems, as well as their algorithms. We then proceed to attack some algorithmic problems in Chapter 5.

Original contributions

During the course of this thesis, we have answered three algorithmic problems, namely the Split → Cluster problem, Well-Partitioned Chordal → Complete Split problem, as well as the Well-Partitioned Chordal → Cluster problem. These results can be found in Chapter 5.

Chapter 2

Preliminaries

Throughout this thesis, we use terminology in accordance with [6].

2.1 Graphs

A *graph* G is a pair of sets (V, E) such that $E \subseteq V \times V$ (that is, 2-element subsets of V). The elements of V are called *vertices*, and the set is referred to as $V(G)$. The elements of E are called *edges*, and the set is referred to as $E(G)$; the sets may simply be referred to as V and E, respectively, when it is clear which graph G is being studied. The number of vertices of a graph G is called its *size* or *order* and is denoted by $|G|$. Graphs may be finite, infinite or uncountable.

A vertex v is *adjacent* to another vertex u where $u, v \in V(G)$ if and only if the element $(u, v) \in E(G)$. If $e = (u, v) \in E(G)$, the vertices u and v are the *endvertices* or *ends* of the edge e.

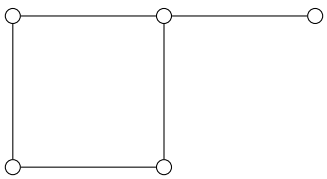


Figure 2.1: A simple graph G

The *neighborhood* (more precisely, the *open neighborhood*) of a vertex v is the set of vertices x such that $(v, x) \in E$, that is, the set of vertices adjacent to v. The *closed neighborhood* of v is equal to $v \cup N(v)$, and is denoted by $N[v]$. The *degree* of a vertex v is equal to $|N(v)|$. Further, $\delta(G) = \min_{v \in V} d(v)$ is the *minimum degree of* G, and $\Delta(G) = \max_{v \in V} d(v)$ is the *maximum degree of* G.

A graph $G' = (V', E')$ is called a *subgraph* of a graph $G = (V, E)$, denoted as $G' \subseteq G$, if $V' \subseteq V$ and $E' \subseteq E$. Furthermore, if $G' \subseteq G$ and $E' = \{(x, y) \mid x, y \in V'\}$, the subgraph G' is called an *induced subgraph* of G. In this regard, we say that G' is the subgraph *induced* by the vertex set V', and denote the same as $G[V']$. We say G' is in G (or G contains G') to signify that G' is a subgraph of G.

A graph is *connected* if, for every pair u, v of vertices, there exists a path joining u and v. If a graph is not connected, it is *disconnected*. Every maximally connected subgraph of a disconnected graph is called a *connected component* (when we use the words minimality or maximality, we are referring to the subgraph relation). A *minimal vertex separator* of two vertices $u, v \in V$ is an inclusion-wise minimal set $S \subset V$ such that, in the graph $G[V - S]$, u and v belong in different connected components of $G[V - S]$ (when dealing with sets, we sometimes denote the subtraction of sets operation "\" simply by the subtraction sign $-$).

The *complement* of the graph $G = (V, E)$ is the graph $\overline{G} = (V, E')$ with $E' = \{(u, v) \in V \times V \mid (u, v) \notin E\}$, that is, the graph on the same vertex set, with the edge set being the set of edges that are not in G.

A *disjoint union* of k graphs $G_1, G_2, \ldots G_k$ with vertex sets $V_i, 1 \leq i \leq k$ and edge sets $E_i, 1 \leq i \leq k$, $k \geq 2$, is the graph $\mathcal{G} = (V_1 \cup V_2 \cdots \cup V_k, E_1 \cup E_2 \cdots \cup E_k)$. Here, each graph is disjoint from every other graph.

Figure 2.2: A graph G and its complement $\overline{G}$

2.2 Characterising Graphs

A *labelling* of a graph is the assignment of labels to the vertices and edges of the graph. A *graph property* is any property of graphs which depends only on the abstract structure of the graph, and is not concerned with the particular labelling of the graph. If a graph G is the same as a graph H (upto labellings) we simply write $G = H$, and state that G is equal to or the same as H. In this thesis, we are only interested in such properties, and we refer to such a graph property simply as property.

A property which is true for every induced subgraph of the graph is called a *hereditary property*. A *graph class* is any property that characterises a set of graphs. For instance, the set of empty graphs is a graph class with the common property of being edgeless. A graph class $\mathcal{C}_1$ is said to be a *subclass* of another graph class $\mathcal{C}_2$ if, for every $G \in \mathcal{C}_1$, it holds that $G \in \mathcal{C}_2$.

A graph G is *self-complementary* if $G = \overline{G}$. We say that a graph class $\mathcal{C}$ is *closed under complements* if $G \in \mathcal{C} \iff \overline{G} \in \mathcal{C}$. Note that, here, we do not require the complement to be the same as the original graph, we only need it to be in the same graph class (that is, satisfy the same property as G).

Some elementary graph classes are enumerated below, with their notation. The definitions of walks, trails and paths are as found in [6].

- A *complete graph* is a graph in which every vertex is adjacent to every other vertex, that is, $E = V \times V$. It is denoted by K_n, where n is the order of the graph.
- A *clique* is any complete subgraph of a graph.
- An *empty graph*, denoted by I_n (n being the order of the graph), is a graph in which no vertex is adjacent to any other vertex, or, in other words, $E = \varnothing$.
- A *cluster graph* is a disjoint union of cliques.
- A *path graph* P_k, (k is the order of the graph) is a graph having a singular path of length k.
- A *cycle* is a non-empty trail in which only the first and last vertices are equal. A *cycle*

graph C_n (n being the order of the graph) is a graph that contains a single cycle or size n. If a graph includes no cycles, it is called *acyclic*

- A *tree* is any connected acyclic graph. A *forest* is a disjoint union of trees.

A *forbidden subgraph* F of G is any graph F such that F is not an induced subgraph of G. A forbidden subgraph of a graph class $\mathcal{C}$ is any graph which is not an induced subgraph for any $G \in \mathcal{C}$. A *forbidden subgraph characterisation* of a graph class $\mathcal{C}$ is a (possibly infinite) set $\mathcal{F}$ of forbidden subgraphs, and we denote the class $\mathcal{C}$ as $\mathcal{F}$-free graphs. Hence, no graph of $\mathcal{G}$ can contain any graph $F \in \mathcal{F}$ as an induced subgraph. For instance, cluster graphs are also defined as P_3-free graphs.

2.3 Parameters

We now introduce some basic concepts related to subsets and partitions of vertices. These concepts not only play a crucial role in characterising various types of graphs, but have various practical implications, and are, thus, extensively investigated from an algorithmic perspective.

2.3.1 Subsets of Vertices

For a graph, $G = (V, E)$, a subset $S \subseteq V$ for which it holds that $G[V - S]$ contains no edges is called the *vertex cover* of a graph. In other words, $\forall\ e \in E,\ \exists\ v \in S$ such that v is an endvertex of e. The smallest such subset of V is called the *minimum vertex cover* of G and the order of this set is denoted by $\tau(G)$.

Any set $I \subseteq V$ such that $G[I]$ contains no edges (i.e., no vertex of I is adjacent to any other vertex of I) is called an *independent set* of the graph. The largest such independent set in G is called the *maximum independent set* of G. The order of the largest independent set is called the *independence number* of G, and is denoted by $\alpha(G)$. Notice that, if S is a vertex cover, $V - S$ will be an independent set (and vice-versa).

A *clique* of a graph is a maximal complete subgraph of G, and the clique (in G) of the

largest order is called the *maximum clique* of G. The order of the largest clique is called the *clique number* of G, and is denoted by $\omega(G)$. We note that the complement of a clique is an independent set (and vice-versa).

2.3.2 Partitions of Vertices

A *proper vertex coloring* of the vertices of G is an assignment of labels $l(v), v \in V$ to vertices of G such that, if $(u, v) \in E$, then $l(u) \neq l(v)$. This implies that no two adjacent vertices can be assigned the same label. The labels used are usually denoted by colors, and we shall do the same. The *chromatic number* of G is the least number of colors required to properly color the vertices of G, and is denoted by $\chi(G)$.

A proper vertex coloring can also be interpreted as a partition of vertices into sets $S_1, S_2, \ldots S_k$ such that, for each $1 \leq i \leq k$, $G[S_i]$ is edgeless. If, on the other hand, we require each partition to induce a clique (that is, each $G[S_i], 1 \leq i \leq k$ is a clique), we arrive at a *clique cover* of the graph. A clique cover that uses the fewest number of cliques possible is called the *minimum clique cover* of G, and is denoted by $k(G)$.

These parameters are not only important from a practical point of view, but also form an important tool for characterising classes of graphs. In this regard, a *perfect graph* is defined as any graph G for which $\chi(H) = \omega(H)$ holds for all induced subgraphs H of G. Perfect graphs are extensively studied, owing to their various structural properties, as well as algorithmic implications.

2.3.3 Treewidth

To define the notion of treewidth, we require the concept of tree decompositions. A thorough treatment of this concept can be found in [7, Chapter 7].

A *tree decomposition* of a graph $G = (V, E)$ is a pair $\mathcal{T} = (T, E_T)$, where each element $T_1, T_2, \ldots T_k \in T$ is a subset of vertices of G (called bags), and $\mathcal{T}$ is a tree such that:

1. $\forall v \in V$, $\exists\ T_i \in T$ such that $v \in T_i$.

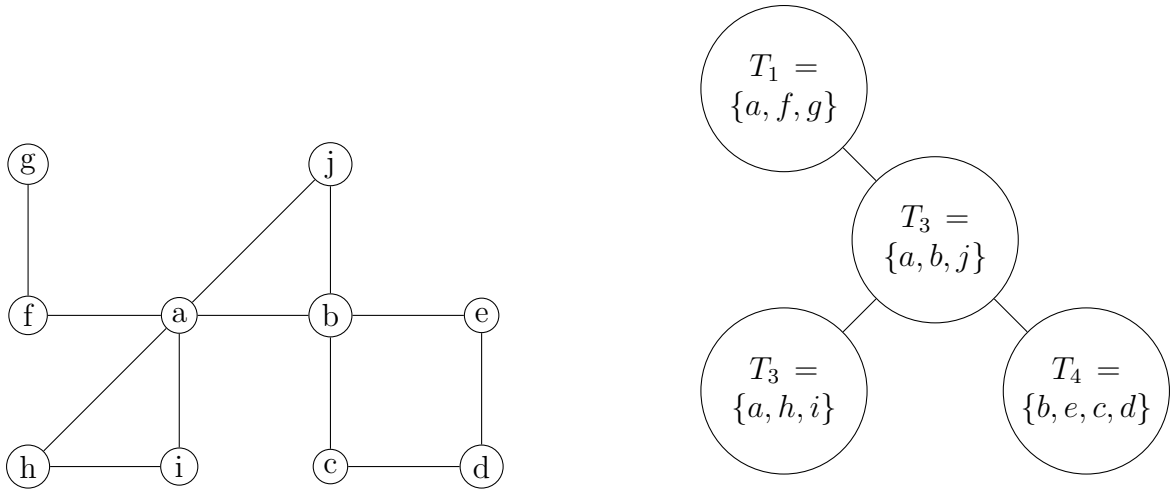


Figure 2.3: A graph G and its tree decomposition $\mathcal{T}$

2. If $(u, v) \in E$, then $\exists\ T_j \in T$ such that both u and v are in T_j.

3. Let $X_v = \{T_i \in T \mid v \in T_i\}$. Then, the graph $\mathcal{T}[X_v]$ induced by the vertex set X_v is a connected subtree of $\mathcal{T}$. This is called the *induced subtree property*.

The tree $\mathcal{T}$ is then called the tree decomposition of G.

We now elucidate an equivalent formulation of tree decompositions, utilizing the *running intersection property*, which states that, if T_i, T_j and T_k are bags, and T_j is on the path from T_i to T_k, then $T_i \cap T_k \subseteq T_j$. On swapping the induced subtree property with the running intersection property (keeping the other conditions as is), we obtain an equivalent formulation of tree decompositions. The proof of equivalence of the 2 formulations is elementary, and we skip the proof for the same.

The *width* of the tree decomposition $\mathcal{T}$ is defined as $w(\mathcal{T}) = \left(\max_{1 \le i \le k} |T_i|\right) - 1$. The *treewidth* of the graph G is $tw(G) = \min_{\mathcal{T}} w(\mathcal{T})$ i.e., the tree decomposition of G with the least width.

Finding the treewidth of an arbitrary graph is computationally hard ([8]), and numerous tools have been developed to get around this intractability. This concept is vital as many problems on graphs have been attacked using the concept of tree decompositions and treewidth, to build efficient algorithms.

2.4 Algorithms

An *algorithm* is any finite sequence of instructions, used to solve a target problem. In what follows, we will give a short introduction to efficiency and time complexity, as well as computational intractability. The following is sourced from [9].

2.4.1 Running time analysis

We are interested in finding the "time" it takes to execute the algorithm, given an input instance. The *running time* of an algorithm is the number of operations performed by the algorithm in the worst-case scenario. The efficiency of an algorithm is defined based on the size of the input. Let f, g be two real-valued functions that are positive for all values of n. Then, $f(n) = O(g(n))$ as $n \to \infty$ if $|f(n)| \leq M \cdot |g(n)|$ $\forall n \geq n_0$ for some $n_0 \in \mathbb{N}$. Thus, if an algorithm has running time $T(n) = c \cdot n^3$, where n is the size of the input and $c \in \mathbb{R}$, we write $T(n) = O(n^3)$, and say that the algorithm has *order* of n^3 complexity. In general, if $T(n) = O(n^c)$, $c \in \mathbb{R}$, the algorithm is said to run in *polynomial time*.

2.4.2 Complexity classes

A *decision problem* is a computational problem that can be posed as a question with a yes or no answer. A decision problem is said to be in the complexity class P if the problem has a polynomial time algorithm.

If a decision problem is such that the solution is verifiable in polynomial time, the problem is said to be in class NP. From here on in, we write that a problem $X \in C$ to mean that X is in the complexity class C.

For decision problems X and Y, a *polynomial-time reduction* from X to Y is an algorithm, computable in polynomial time, with:

- Input: I_X, an instance of problem X
- Output: I_Y, an instance of problem Y.

with the requirement that I_Y is a YES instance of Y if and only if I_X is a YES instance of X.

A problem Y is an *NP-complete problem* if:

- $Y \in NP$
- $\forall X \in NP$, we have that $X \leq_P Y$.

Thus, informally, we might say that the NP-complete problems are the "hardest" problems of the NP class.

This concludes our exposition on some elementary properties required to the remainder of the thesis.

Chapter 3

Chordal Graphs and Their Subclasses

Chordal graphs are a highly studied graph class, owing to their rich structural as well as algorithmic properties. One can think of chordal graphs as a relaxation of trees. Observe that trees forbid any cycle, whereas chordal graphs are graphs that permit the cycle C_3, and forbid all cycles of greater length. We will cover some basic properties of chordal graphs, before moving to some of their subclasses.

3.1 Chordal Graphs

Here, we introduce the chordal graph class, and mention some basic definitions useful for characterising chordal graphs. The following is sourced from [1, Chapter 4].

Definition 3.1.1. *An edge of a graph is a* ***chord*** *if it connects two non-adjacent vertices of a cycle of the graph.*

Definition 3.1.2. *A graph G is called a* ***chordal graph*** *if every induced cycle of length greater than 3 possesses a chord. If a graph is chordal, we say it satisfies the chordality property.*

Definition 3.1.3. *A vertex v in G for which $N[v]$ is a complete subgraph, is called a* ***simplicial vertex****.*

Definition 3.1.4. *A* ***perfect elimination ordering*** *of the vertices is an ordering $\sigma =$*

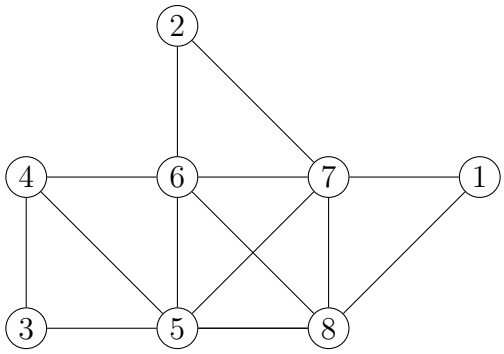


Figure 3.1: A chordal graph G, with its perfect elimination ordering

$(v_1, v_2, \ldots v_n)$ *such that* v_i *is a simplicial vertex of* $G[\{v_i, v_{i+1}, \ldots v_n\}]$. *If the vertices of* G *have a perfect elimination ordering, we say that* G *has a perfect elimination ordering.*

3.1.1 Basic Properties

We now go over some of the most important characteristics of chordal graphs and characterise this class of graphs.

Theorem 3.1.1. *The chordality property of graphs is hereditary.*

Proof. Let $G[V'] = G'$ be any induced subgraph of G (where $V' \subseteq V$). Assume that G' contains a chordless cycle $C = [v_1, v_2, \ldots v_k]$. Since G' is an induced subgraph of G, C is also a chordless cycle of G (as induced subgraphs preserve edges between all the vertices present in V'). We arrive at a contradiction. Hence, $G[V']$ cannot have a chordless cycle, indicating that a chordal graph's induced subgraphs are chordal. □

Theorem 3.1.2. *[1, Chapter 4] Let* $G = (V, E)$ *be a graph. Then, the following statements are equivalent:*

1. *G satisfies chordality.*
2. *Every minimal vertex separator of G induces a complete subgraph of G.*

Proof. ((2) $\implies$ (1)) Let $[x, v, y, v_1, v_2, \ldots v_k, x]$ be a simple cycle of G. Then, every minimal vertex separator of $x - y$ contains the vertices v, as well as v_i, for some $i \in [1, k]$. This implies

$(v, v_i) \in E$, as minimal vertex separators induce a complete subgraph of G. Observe that (v, v_i) is a chord of the cycle.

((1) $\implies$ (2)) Let S be a minimal $x - y$ separator of G, and let G_x, G_y be the connected components of $G[V - S]$ containing the vertices x and y, respectively. As S is minimal, each vertex $s \in S$ has a neighbor in both G_x as well as G_y. Hence, for any $s, s' \in S$, there exist smallest length paths $[s, x_1, x_2, \ldots x_k, s']$ and $[s', y_1, y_2, \ldots y_j, s]$, where each $x_i \in G_x$ and each $y_i \in G_y$. Now, observe that $[s, x_1, x_2, \ldots x_k, s', y_1, y_2, \ldots y_j, s]$ is a cycle of G of length atleast 4, and hence it has a chord. Since these paths are minimal, none of $(x_i, x_{i+2}) \in E$ (where $i \in [1, k-2]$). Similarly, none of $(y_i, y_{i+2}) \in E$ (where $i \in [1, j-2]$). Therefore, the only possible chords are (x, y) or (s, s'). However, if $(x, y) \in E$, it violates the definition of vertex separator. Therefore, we have $(s, s') \in E$. □

We require the following lemma to demonstrate a characterisation of chordal graphs.

Lemma 3.1.3. *[1, Chapter 4] If a chordal graph G is not a complete graph, then it has two non-adjacent simplicial vertices.*

Proof. We proceed by induction. Assume the statement is true for all graphs with fewer vertices than G. Let S be the minimal vertex separator of $x - y$, and $G[V_x]$ and $G[V_y]$ be the connected components of $G[V - S]$ containing x and y, respectively. Since $G[S \cup V_x]$ is chordal and has fewer vertices than G, it has 2 non-adjacent simplicial vertices, with at least one of them (say v_1) in $G[V_x]$ (as $G[S]$ is complete by Theorem 3.1.2). Moreover, since $N[V_x] \subseteq S \cup V_x$, v_1 is simplicial in all of G. A similar argument works on the graph $G[S \cup V_y]$, and we can find a simplicial vertex v_2 of G. Hence, we have found 2 non-adjacent simplicial vertices in the chordal graph G. □

Theorem 3.1.4. *[1, Chapter 4] Chordal graphs are characterised by the perfect elimination ordering property of vertices. In fact, this ordering is not unique, and any simplicial vertex can be the first vertex of the ordering.*

Proof. Assume G is chordal, and the theorem holds true for all chordal graphs with fewer vertices than G. By Lemma 3.1.3, we can find a simplicial vertex v_1 of G. Now, observe that $G[V - v_1]$ is chordal, and hence has a perfect elimination ordering σ'. Append v_1 as a prefix to σ' to obtain a perfect elimination ordering σ of G.

Now, let G have a perfect elimination ordering σ, and C be a cycle in G. Additionally, let v be the vertex in C such that v has the smallest index in σ. Since v at least 2 neighbors in C, the subsequent simpliciality of v ensures that there is a chord in C, rendering the graph G chordal. □

Having covered some basic structural properties of chordal graphs, we now move to one of the most important properties of chordal graphs, namely that chordal graphs are perfect.

Theorem 3.1.5. *[1, Chapter 4] All chordal graphs are perfect.*

To prove this theorem, we need the following lemma.

Lemma 3.1.6. *[1, Chapter 4] Let S be any (not necessarily minimal) vertex separator of any connected graph $G = (V, E)$, and $G_{C_1}, G_{C_2} \dots G_{C_t}$ be the connected components of $G[V - S]$. If S is a clique of G (which need not be maximal), then:*

$$\chi(G) = \max_i \chi(G[S] \cup G_{C_i}) \quad \textit{and} \quad \omega(G) = \max_i \omega(G[S] \cup G_{C_i})$$

Proof. It is trivial to note that $\chi(G) \geq \chi(G[S] \cup G_{C_i})\ \forall i$. Thus, $\chi(G) \geq \max_i \chi(G[S] \cup G_{C_i})$. We claim that G can be colored with exactly $k = \max_i \chi(G[S] \cup G_{C_i})$ colors. We first assign colors to each vertex of S (observe that each vertex will have a unique color as S is a clique). Now, since each G_{C_i} is separated from G_{C_j} for $i \neq j$, we can independently extend the coloring of S to each $G[S] \cup G_{C_i}$. Following this procedure, we obtain a coloring of G with exactly k colors, as $\exists\ i \in [1, t]$ such that $\chi(G[S] \cup G_{C_i}) = k$.
To prove our second assertion, we, once again, note that $\omega(G) \geq \omega(G[S] \cup G_{C_i})\ \forall i$. Thus, $\omega(G) \geq \max_i \omega(G[S] \cup G_{C_i})$. Let K be the maximum clique of G, implying $\omega(G) = |K|$. As K is a clique, no 2 vertices of K can lie in different components of $G[V - S]$ (as every vertex of K is adjacent to every other vertex of K). Thus, $\exists j \in [1, t]$ such that $\omega(G) \geq \omega(G[S] \cup G_{C_j})$. Thus, we have

$$\omega(G[S] \cup G_{C_j}) \geq |K| \qquad \text{from above assertion} \tag{3.1}$$

$$|K| = \omega(G) \qquad \text{by definition of } \omega(G) \tag{3.2}$$

$$\therefore \omega(G[S] \cup G_{C_j}) \geq \omega(G) \qquad \text{from 3.1 and 3.2} \tag{3.3}$$

We already know that $\omega(G) \geq \omega(G[S] \cup G_{C_i})\ \forall i$. Therefore, we have $\omega(G) = \max_i \omega(G[S] \cup G_{C_i})$. □

The corollary follows from the previous theorem.

Corollary 3.1.7. *[1, Chapter 4] Let S be any (not necessarily minimal) vertex separator of any connected graph $G = (V, E)$, and $G_{C_1}, G_{C_2} \ldots G_{C_t}$ be the connected components of $G[V - S]$. If S is a (not necessarily maximal) clique of G, and $G[S] \cup G_{C_i}$ is perfect $\forall\ i \in [1, t]$, then G is perfect.*

Proof. Since $G[S] \cup G_{C_i}$ is perfect $\forall\ i \in [1, t]$, we have $\max_i \chi(G[S] \cup G_{C_i}) = \max_i \omega(G[S] \cup G_{C_i})$. Thus, by Lemma 3.1.6, we have

$$\chi(G) = \max_i \chi(G[S] \cup G_{C_i}) = \max_i \omega(G[S] \cup G_{C_i}) = \omega(G)$$

which proves that G is perfect. □

We are now equipped to prove Theorem 3.1.5.

Proof of Theorem 3.1.5

Proof. We will proceed by induction on the order of the graph. Assume G is chordal, and the theorem holds true for all graphs with fewer vertices than G. If G is complete, we are done. Assume, then, that G is not complete (but is connected). Let S be any minimal vertex separator of G. By Theorem 3.1.2, S is complete. Consider the connected components $G[S] \cup G_{C_i}, (1 \leq i \leq t)$ of $G[V - S]$. Observe that each $G[S] \cup G_{C_i} (1 \leq i \leq t)$ is perfect (due to Theorem 3.1.1 and induction hypothesis). Therefore, by Corollary 3.1.7, G is perfect. □

3.1.2 Recognising Chordal Graphs

Now that we have defined and studied some basic properties of chordal graphs, we learn how to recognise and verify whether a given graph G is chordal. We exploit the fact that (non-complete) chordal graphs have at least 2 non-adjacent simplicial vertices, to build an

easy and efficient recognition algorithm for chordal graphs. We first present a pseudocode of the algorithm, and then argue for its correctness. The algorithm is named Lexicographic Breadth-First Search, and it is a modification of the classical BFS algorithm ([1, Chapter 4]). We define the following notations required for the algorithm:

- Let σ_G be the resultant ordering on running Algorithm 1.
- We assign a set 'label', denoted by $l(v)$, to each vertex $v \in V$. The largest value of $l(v)$ (for a given v) is $m(l(v))$.
- We call a vertex 'un-numbered' if it has not yet been assigned a number in the ordering σ_G.
- The set of un-numbered vertices is $U(G)$. The un-numbered vertex with the largest $m(l(v))$ is $f(U)$.

Algorithm 1 LexBFS algorithm

Input: $G = (V, E)$
Output: σ_G ▷ Finds a LexBFS ordering σ_G of G

```
1: procedure ORDERING((G = (V, E))
2:     for all i = n to 1, Step = -1 do
3:         v ← f(U)
4:         σ_G[i] ← v                ▷ This assigns to v the number i
5:         for all v ∈ U(G) ∩ N(v) do
6:             l(v) ← l(v) ∪ {i}
7:         end for
8:     end for
9:     return σ_G
10: end procedure
```

Theorem 3.1.8. *[1, Chapter 4] For any chordal graph $G = (V, E)$, the Algorithm 1 produces a perfect elimination ordering.*

From Theorem 3.1.4, we know that, if a perfect elimination ordering exists, then the graph is chordal. Hence, the above theorem characterises chordal graphs with respect to the LexBFS algorithm.

To prove this theorem, we require the following inferences. Let $l_i(v)$ be the label of v when the i^{th} vertex is numbered, and $\sigma = \sigma_G$

P1. If $j \leq i, m(l_i(v)) \leq m(l_j(v))$ i.e, the appended values in $l(v)$ are successively decreasing.

P2. If $j < i$ and $m(l_i(u)) < m(l_i(v))$ (where $u, v \in V$), then $m(l_j(u)) < m(l_j(v))$.

P3. If $\sigma^{-1}(u) < \sigma^{-1}(v) < \sigma^{-1}(w)$ and $w \in N(u) \setminus N(v)$, then there exists $x \in N(v) \setminus N(u)$ such that $\sigma^{-1}(w) < \sigma^{-1}(x)$.

We are now in a position to prove Theorem 3.1.8.

Proof of Theorem 3.1.8

Proof. Let us proceed by induction on the size of V. The theorem is trivial for $|V| = 1$. Assume that all graphs with fewer vertices than G satisfy the theorem, and let σ be the output ordering of Algorithm 1 given a chordal graph G as input (G has order n). By induction, it is enough to demonstrate that $v = \sigma(1)$ is a simplicial vertex of G (for $u, v \in V$, we say u is larger than v if $\sigma^{-1}(v) < \sigma^{-1}(u)$).
Assume this is not the case, and choose $v_1, v_2 \in N(v)$ with $(v_1, v_2) \notin E$ such that (v_2) is as large as possible. We construct the following sequence of vertices inductively, assuming that we have been given vertices $v_1, v_2, \ldots v_m$ such that the following holds: $\forall i, j > 0$

1. $v_0 = v, v_{-1} = v_1$
2. $(v, v_i) \in E \iff i \leq 2$
3. $(v_i, v_j) \in E \iff |i - j| = 2$
4. $\sigma^{-1}(v_1) < \sigma^{-1}(v_2) < \cdots < \sigma^{-1}(v_m)$
5. v_j is the vertex with the largest $\sigma^{-1}(v_j)$ such that $(v_{j-2}, v_j) \in E$ but $(v_{j-3}, v_j) \notin E$

For $m = 2$, we have already constructed the vertices. We observe that v_{m-2}, v_{m-1}, v_m satisfy (P3) as u, v and w respectively. Choose v_{m+1} to be the largest vertex larger than v_m which is adjacent to v_{m-1} but not adjacent to v_{m-2}. If v_{m+1} was adjacent to v_{m-3}, then apply (P3) to the vertices $v_{m-3}, v_{m-2}, v_{m+1}$ to obtain a vertex v' larger than v_{m+1} (and thus larger than v_m) adjacent to v_{m-2} but not to v_{m-3}. This contradicts the maximality (w.r.t σ) of v_m in 5. Hence, $(v_{m+1}, v_{m-3}) \notin E$. Thus, it follows from 2, 3 and chordality of G that $(v_i, v_{m+1}) \notin E$

for $i = 0, 1, \ldots m-4, m$.
We notice that this inductive procedure continues without end, however, the graph contains only n vertices. We arrive at a contradiction, and thus infer that the vertex v is simplicial, proving the theorem. □

3.1.3 Basic Algorithms

Chordal graphs are extensively studied, especially from an algorithmic point of view, as various classically NP-Complete problems (such as Max Clique, Max Independent Set and Min Vertex Cover) are efficiently solvable on chordal graphs. In this section, we review some algorithms for solving the above problems on chordal graphs. We can appreciate the algorithmic utility of the perfect elimination ordering by observing that these algorithms are built by exploiting this very characteristic of chordal graphs.

Maximum Clique

Lemma 3.1.9. *[1, Chapter 4] For a chordal graph $G = (V, E)$ with perfect elimination ordering σ, every maximal clique is of the form $\{v\} \cup X_v$, where $X_v = \{u \in N(v) | \sigma^{-1}(v) < \sigma^{-1}(u)\}$.*

Proof. By definition, $\{v\} \cup X_v$ is complete $\forall v \in V$. Now, let y be the earliest vertex in σ in some maximal clique C of G. Then, $C = \{y\} \cup X_y$. If not, then either $C \subset X_y$ or $X_y \subset C$. If $C \subset X_y$, then C is not maximal. If $X_y \subset C$, then, since y is the earliest vertex of σ in C, X_y can be extended to C. In both cases, we arrive at a contradiction. Hence, each maximal clique is of the form $\{v\} \cup X_v, v \in V$. □

We now illustrate a basic algorithm to list all maximal cliques of the chordal graph G, thereby also finding the maximum clique of G.

Given a chordal graph $G = (V, E)$ of order $|V| = n$ and perfect elimination ordering σ, we define the following:

- For each $v \in V$, $X_v = \{u \in N(v) | \sigma^{-1}(v) < \sigma^{-1}(u)\}$, and $Y_v = \{v\} \cup X_v$.

- $\sigma[i]$ denotes the i^{th} element in the ordering σ.
- MC_G is the set of all maximal cliques of G.

Algorithm 2 Maximal Cliques algorithm

Input: $G = (v, E, \sigma)$ ▷ Given a chordal graph with simplicial ordering
Output: $MC_G, \omega(G)$ ▷ Returns all maximal cliques, as well as $\omega(G)$

```
1: procedure CLIQUE((G = (V, E, σ))
2:     for all i = 1 to n do
3:         v ← σ[i]
4:         if Y_v is maximal then                ▷ All maximal cliques are of this type
5:             MC_G ← MC_G ∪ {Y_v}
6:         end if
7:     end for
8:     ω(G) = |max{MC_G}|
9:     return MC_G, ω(G)
10: end procedure
```

Theorem 3.1.10. *Algorithm 2 returns all maximal cliques of any chordal graph G (of order n) in polynomial time.*

Proof. Since all maximal cliques are of the type $Y_v, v \in V$ (owing to Lemma 3.1.9) there can be at most n maximal cliques of any chordal graph G. Hence, it suffices to check each set of this type for maximality. Checking a set for maximality takes at most $O(n)$ time, and since there exist at most n sets of the type Y_v, our algorithm runs in time at most $O(n^{O(1)})$. □

Since chordal graphs are perfect, $\omega(G) = \chi(G)$, and, hence, the above algorithm also finds the chromatic number of the graph.

Maximum Independent Set, Minimum Clique Cover

Our algorithm for resolving the maximum independent set problem on chordal graphs is built upon the ensuing theorems.

Theorem 3.1.11. *[10] The perfect graph class is closed under complements.*

The above theorem is also known as the Weak Perfect Graph Theorem, and it was proved by Lovasz in 1972. Since the proof is not relevant here, we omit it and refer the reader to [10] for the same. This theorem has the following corollary as a result.

Corollary 3.1.12. *If $G = (V, E)$ is perfect, the size of the minimum clique cover and maximum independent set is equal, i.e. $k(G) = \alpha(G)$.*

Proof. Since G is perfect, we have $\chi(G) = \omega(G)$. We make the following observations:

1. Since any clique of $\overline{G}$ is an independent set of G, we have that $\omega(\overline{G}) = \alpha(G)$.
2. A proper coloring of V partitions it into $\chi(G)$ parts $V_1, V_2, \ldots V_{\chi(G)}$, each of which is an independent set, by definition. Hence, in $\overline{G}$, each part V_i is a clique. Thus, we have that $V_1, V_2, \ldots V_{\chi(G)}$ is a clique cover of G, and we obtain $\chi(\overline{G}) = k(G)$.

From Theorem 3.1.11, $\overline{G}$ is perfect. Thus, we have

$$\begin{aligned} \omega(\overline{G}) &= \chi(\overline{G}) && \text{from above assertion} \\ \omega(\overline{G}) &= \alpha(G) && \text{from (1)} \\ k(G) &= \chi(\overline{G}) && \text{from (2)} \\ \therefore \quad k(G) &= \alpha(G) \end{aligned}$$

Hence, for any perfect graph, the size of the maximum independent set and minimum clique cover are the same. □

Now, to find the independent set of a chordal graph $G = (V, E)$ with the perfect elimination ordering σ, we define the following sequence of vertices inductively:

1. $y_1 = \sigma(1)$
2. y_i is the first vertex in σ following y_{i-1} which is not in any $X_{y_1} \cup X_{y_2} \cup \ldots X_{y_{i-1}}$.

Here, $\sigma(i)$ denotes the ith element in the ordering σ, and $X_v = \{u \in N(v) | \sigma^{-1}(v) < \sigma^{-1}(u)\}$. Observe that $V = \{y_1, y_2, \ldots y_t\} \cup X_{y_1} \cup X_{y_2} \cup \ldots X_{y_t}$.

We are now equipped to prove our main theorem. We retain the same notation as introduced above.

Theorem 3.1.13. *[1, Chapter 4] The set $\{y_1, y_2, \dots y_t\}$ is a maximum independent set of G, and the collection of sets $Y_i = \{y_i\} \cup X_{y_i}, 1 \leq i \leq t$ comprises a minimum clique cover of the chordal graph $G = (V, E)$.*

Proof. Since no y_i is adjacent to any y_j for all $1 \leq i, j \leq t, \{y_1, y_2, \dots y_t\}$ is an independent set of G. Thus, $\alpha(G) \geq t$.
For all $1 \leq i \leq t$, Y_i is a clique (by definition of X_{y_i}). Additionally, $\bigcup_i Y_i = V$. To ensure that this forms a partition, notice that if any 2 sets Y_i, Y_j (for $1 \leq i, j \leq t$) have common vertices, we can simply place the common vertices in any one of the sets as both of the sets will induce cliques. Hence, $k(G) \leq t$.
Combining the above 2 assertions with Corollary 3.1.12, we have that $t \leq \alpha(G) = k(G) \leq t \implies \alpha(G) = k(G) = t$. □

Below, we elucidate a basic algorithm which finds the maximum independent set of a chordal graph G, based on the theorem above.

Algorithm 3 Independent Set algorithm

Input: $G = (v, E, \sigma)$ ▷ Given a chordal graph with simplicial ordering
Output: $\alpha(G)$ ▷ Returns independent set of chordal graph

```
1: procedure IndSet((G = (V, E, σ))
2:     I ← ∅                                   ▷ I is the required maximum independent set
3:     for all i = 1 to n do
4:         v ← σ[i]
5:         if v ∉ ⋃_{u∈I} X_u then              ▷ Ensuring v is independent to all i ∈ I
6:             I ← I ∪ {v}
7:         end if
8:     end for
9:     α(G) = |I|
10:    return α(G)
11: end procedure
```

Illustration 1. *For the chordal graph G shown in Figure 3.2, we demonstrate the algorithms to find the maximum clique and maximum independent set.*

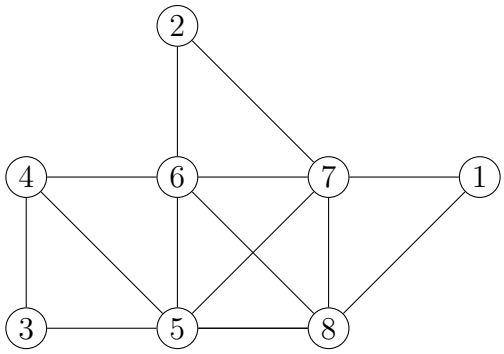


Figure 3.2: A chordal graph G, with its perfect elimination ordering

Maximum Clique

v	Y_v
1	$\{1, 7, 8\}$
2	$\{2, 6, 7\}$
3	$\{3, 4, 5\}$
4	$\{4, 5, 6\}$
5	$\{5, 6, 7, 8\}$
6	$\{6, 7, 8\}$
7	$\{7, 8\}$
8	$\{8\}$

The cliques corresponding to the colored rows are maximal. Thus, our set of maximal cliques is

$$MC_G = \{Y_1, Y_2, Y_3, Y_4, Y_5\}$$

with $|Y_5| = 4$ being the maximum. Thus, $\omega(G) = 4$.

Maximum Independent Set

v	I
1	$\{1\}$
2	$\{1, 2\}$
3	$\{1, 2, 3\}$
4	$\{1, 2, 3\}$
5	$\{1, 2, 3\}$
6	$\{1, 2, 3\}$
7	$\{1, 2, 3\}$
8	$\{1, 2, 3\}$

We update I whenever some $v_i \notin \bigcup_{u \in I} X_u$. The iterations for which I are updated are colored. We have

$$I = \{1, 2, 3\}$$

Thus, $\alpha(G) = 3$.

3.1.4 Clique Trees

In this part of the chapter, we cover the concept of clique graphs and clique trees of chordal graphs, which are an important construction having several structural and algorithmic conse-

quences. This concept was studied in detail in [11], from which we have sourced the following content.

Definition 3.1.5. *For a chordal graph $G = (V, E)$, the **Weighted Clique-Intersection Graph** $C(G) = (\mathcal{K}_G, E_K, w)$ with $w = E_K \to \mathbb{N}$ is defined as follows:*

1. *$\mathcal{K}_G$ is the set of maximal cliques of G.*
2. *For 2 maximal cliques $K_1, K_2 \in \mathcal{K}_G$, $(K_1, K_2) \in E_K \iff K_1 \cap K_2 \neq \varnothing$.*
3. *For $e = (K_1, K_2) \in E_K, w(e) = |K_1 \cap K_2|$.*

We now introduce the "clique-intersection" property, and make the connection between this property and tree decompositions. As a consequence of this result, we will also arrive at an efficient algorithm to find the tree decomposition of a chordal graph.

Definition 3.1.6. *For a graph $G = (V, E)$, a tree $\mathcal{T}_G = (\mathcal{K}_G, E_\mathcal{T})$ satisfies the **Clique Intersection property** if every vertex common to K and K' is present in every clique on the path from K to K', in the tree $\mathcal{T}$. If a graph G has such a tree associated with it, we call the tree $\mathcal{T}$ the clique-intersection tree of G.*

The significance of the above property is appreciated by the fact that chordal graphs can be completely characterised by this property. The following theorem makes this statement precise and gives a proof of the claim.

Theorem 3.1.14. *[12] Every connected chordal graph has an associated tree $\mathcal{T} = (\mathcal{K}_G, E_\mathcal{T})$ satisfying the clique intersection property. Furthermore, clique-intersection trees exist only for chordal graphs.*

To prove the above theorem, we need the following lemma.

Lemma 3.1.15. *[12] For a chordal graph $G = (V, E)$, a vertex $v \in V$ is simplicial if and only if it belongs to exactly 1 maximal clique.*

Proof. If a vertex $v \in V$ is simplicial, it belongs to the maximal clique $N[v]$. Let us say it belongs to another maximal clique C in G, then every vertex of C has to be adjacent to every vertex of $N(v)$ (as the vertex v is simplicial). This is a contradiction, as we assumed

that both $N[v]$ and C are maximal cliques.

Now, if a vertex belongs to exactly 1 maximal clique, then every neighbor of v is adjacent to every other neighbor of v. Hence, $N[v]$ is a clique, implying that v is simplicial in G. □

We move to the proof of Theorem 3.1.14.

Proof of Theorem 3.1.14

Proof. First, we prove that there exists such a tree for every chordal graph.

We will use induction on the number of vertices of G to prove the statement.
Let G be a chordal graph on n vertices, and the theorem holds true for all chordal graphs with fewer vertices than n. Now, by Lemma 3.1.3, G has a simplicial vertex v. Since v is simplicial, it belongs to exactly 1 maximal clique, say K (by Lemma 3.1.15). By the induction hypothesis, for $G' = G \setminus \{v\}$, there exists a clique-intersection tree. Call this tree $\mathcal{T}' = (K_{G'}, E_{\mathcal{T}'})$. We have 2 cases. Either $K' = K \setminus \{v\}$ is a maximal clique of G', or $\exists\ K'' \in K_{G'}$ such that $K' \subset K''$.
If K' is a maximal clique of G', then we can construct a tree $\mathcal{T}$ whose vertex set is obtained by replacing K' in $\mathcal{T}'$ with K. Since v only belongs to K in G, every other maximal clique remains the same in G'. In other words, $K_{G'} = (\mathcal{K}_G \cup \{K'\}) \setminus \{K\}$. Additionally, the intersection of any 2 maximal cliques in G cannot contain v (due to Lemma 3.1.15). Since the tree $\mathcal{T}'$ obeys the clique-intersection property, so does the tree $\mathcal{T}$.
Now, say $\exists\ K'' \in K_{G'}$ such that $K' \subset K''$. Since $n \geq 2$, we know that $K' \neq \varnothing$. By Lemma 3.1.15, v belongs to only one maximal clique K in G. Hence, in G', no other clique except K is affected. In other words, we have that $K_{G'} = \mathcal{K}_G \setminus \{K\}$. We construct the tree $\mathcal{T}$, whose vertex set is $\mathcal{K}_G$, by adding the vertex K and the edge (K, K'') to $\mathcal{T}'$. Observe that $K \cap K'' = K' \subset K''$. For any $S \in K_{G'}$, the path from S to K passes through K'', and we have that $S \cap K = S \subset K''$. Hence, every clique on this path contains $K \cap S = K'' \cap S$. For any other pair of cliques $S, S' \in K_{G'}$, we know that the clique-intersection property already holds.

Hence, we obtain a tree $\mathcal{T}$ which satisfies the clique-intersection property.

Now, we prove that, if there is a tree associated with the maximal cliques of the graph G satisfying the clique-intersection property, then the graph G is chordal.

Proceeding by induction, let G have n vertices and a clique-intersection tree $\mathcal{T} = (\mathcal{K}_G, E_{\mathcal{T}})$.The claim holds trivially for $n = 1$. Assume that the claim holds true for all graphs with fewer vertices than G.
Let C be a node of degree 1 in $\mathcal{T}$ and P the sole parent of C in $\mathcal{T}$. Since C, P are maximal by the definition of the tree $\mathcal{T}$, $\exists\ v \in C \setminus P$. Additionally, we claim that C is the unique maximal clique to which v belongs. If this were not the case, and v belongs to some $K \in \mathcal{K}_G$, then v exists in every clique on the path from K to C. Since P is the parent of C, we have that $v \in P$, which violates our assumption. Therefore, C is the unique maximal clique to which v belongs. Now, by Lemma 3.1.15, v is a simplicial vertex of G. Take the graph $G' = G \setminus \{v\}$, and let $C' = C \setminus \{v\}$. Furthermore, let $\mathcal{T}'$ the tree associated with G'. Then, either $C' \subset P$, or $C' \not\subset P$. If $C' \not\subset P$, then the $\mathcal{T}'$ is obtained by simply replacing C with C'. If, on the other hand, $C' \subset P$, modify the tree $\mathcal{T}$, to obtain $\mathcal{T}'$ as follows: remove the vertex C, thereby removing the edge (C, P) as well. In both of these cases, the clique-intersection property is satisfied by the resultant tree $\mathcal{T}'$. Hence, the graph G' is chordal, using the induction assumption. By Theorem 3.1.4, G' has a perfect elimination ordering σ'. Since $v \in G$ is a simplicial vertex of G, append v as a prefix to σ' to obtain a perfect elimination ordering σ of G, implying that G is chordal, completing the proof. □

A *spanning tree* of an edge-weighted graph $G = (V, E, w)$ is a subset of edges E_T of G such that the graph with the vertex set V and edge set E_T is a connected tree of G. Furthermore the weight of the tree is $wt(T) = \sum_{e \in E_T} w(e)$. The spanning tree with the largest weight value is called the *maximum spanning tree* (or MST) of G.

Recall the definition of the Weighted clique-intersection graph from 3.1.5. We prove that, given a chordal graph G, one can obtain a clique intersection tree by finding the MST of the weighted clique-intersection graph $\mathcal{W}_G$. In this regard, let Υ^G_{mst} be the set of all maximum spanning trees (MST) of $\mathcal{W}_G$, and Υ^G_{ct} be the set of all clique-intersection trees of G.

Given an edge $e = (K, K')$ in a tree $\mathcal{T}_{mst} \in \Upsilon^G_{mst}$, its removal from the tree results in the partition of vertices of $\mathcal{T}_{mst}$ into 2 different sets. Call these sets V_1 and V_2 respectively. The *basic cutset of the edge* e is the set of edges such that they have 1 endpoint each in V_1 and V_2 (in the graph $\mathcal{W}_G$). To demonstrate our theorem, we also require the following characteristic

of maximum spanning trees.

Lemma 3.1.16. *[12] Given a graph $G = (V, E, w)$, where w is the weight function $w : E \to \mathbb{R}$, a tree $T = (V, E_T, w)$ is the maximum spanning tree of G if and only if the weight of every edge on the path from u to v in T is no less than $w((u, v))$, for every pair of vertices $u, v \in V$ such that $(u, v) \notin E_T$.*

Proof. First, assume that T is a MST, and that there exists an edge (u, v) such that $\exists\ (a, b)$ on the path from u to v such that $w((u, v)) > w((a, b))$. Consider the cycle C formed by taking the edges (u, v), as well as the path P from u to v in T. Removing the edge (a, b) from T and replacing it with (u, v) results in a new connected tree T' whose total edge weight is higher than that of T, as $w((u, v)) > w((a, b))$. This contradicts the fact that T is a MST, rendering our assumption incorrect.
Now, let us assume the existence of a spanning tree T such that for each $(u, v) \notin E_T, w(u, v) \leq w(e)$ for every edge e on the path from u to v in T. If T is not a MST, then there exists some $(u, v) \notin E_T$ such that replacing some $(a, b) \in E_T$ with (u, v) results in a spanning tree T' having higher edge weight. Observe that $[u, v, \ldots a, b, \ldots u]$ forms a cycle of G, as, if it did not, then the connectivity of a and b in T' is violated. However, this means that there is an edge $(u, v) \notin E_T$ such that for some edge (a, b) on the path from u to v in T, we have $w((u, v)) > w((a, b))$, which contradicts our initial assumption. Therefore, T is a maximum spanning tree. □

Theorem 3.1.17. *[1, Chapter 4] Given a connected chordal graph G, it holds that $\Upsilon^G_{mst} = \Upsilon^G_{ct}$.*

Proof. First, let us show that $\Upsilon^G_{ct} \subseteq \Upsilon^G_{mst}$. Let $\mathcal{T}^G_{ct} \in \Upsilon^G_{ct}$. Take any 2 cliques $K, K' \in \mathcal{K}_G$, such that $(K, K') \notin \mathcal{T}^G_{ct}$. Consider the cycle formed by the edged (K, K') as well as all the edges in the path P from K to K' in $\mathcal{T}^G_{ct}$. Every edge in P contains the intersection $|K \cap K'|$, due to the clique-intersection property of $\mathcal{T}^G_{ct}$. Hence, every edge in P has weight at least $|K \cap K'|$, which is exactly the weight of the edge (K, K'). Hence, by Lemma 3.1.16, we infer that $\mathcal{T}^G_{ct} \in \Upsilon^G_{mst}$, proving the claim that $\Upsilon^G_{ct} \subseteq \Upsilon^G_{mst}$ (3.4).
To prove $\Upsilon^G_{mst} \subseteq \Upsilon^G_{ct}$, start with some $\mathcal{T}^G_{mst} \in \Upsilon^G_{mst}$. Choose the tree $\mathcal{T}_{ct}$ such that it has the highest number of edges in common with $\mathcal{T}^G_{mst}$. Now, consider an edge $(K_1, K_2) \in \mathcal{T}^G_{mst}$ such that $(K_1, K_2) \notin \mathcal{T}^G_{ct}$.Consider the basic cutset $CS_{(K_1,K_2)}$ of the edge (K_1, K_2) (in $\mathcal{T}^G_{mst}$). Since $\mathcal{T}^G_{ct}$ is a tree, we obtain a cycle on adding the edge (K_1, K_2) to it. Let this cycle be

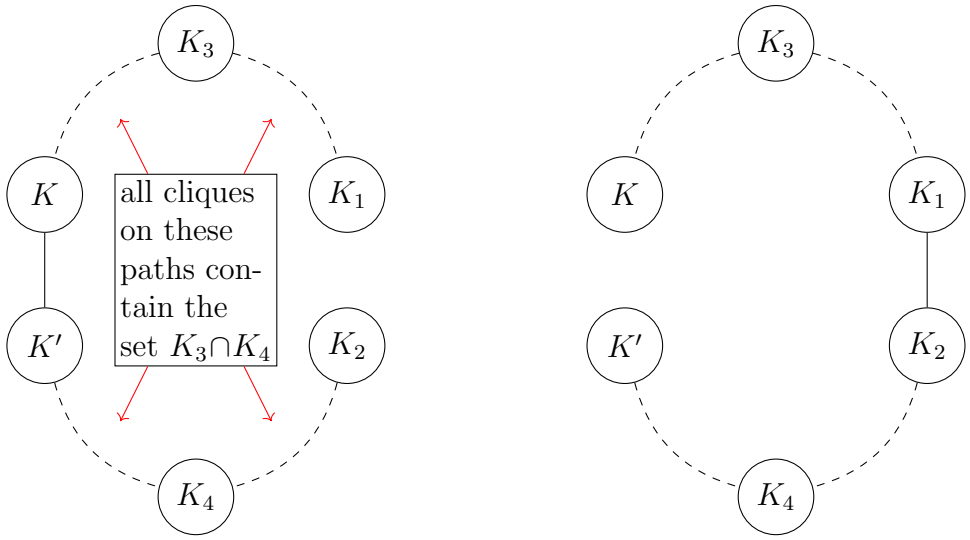
C. Observe that $CS_{(K_1,K_2)}$ contains atleast one edge from this cycle C (as $CS_{(K_1,K_2)}$ is a cutset). Let this edge be $(K, K') \neq (K_1, K_2)$. Due to the definition of the clique-intersection tree $\mathcal{T}_{ct}^G$, we have that $K_1 \cap K_2 \subseteq K \cap K'$. If $K_1 \cap K_2 \subset K \cap K'$, then replacing (K_1, K_2) with (K, K') gives us a spanning tree of higher weight than $\mathcal{T}_{mst}$, contradicting its definition. Hence, $K_1 \cap K_2 = K \cap K'$. Now, replace (K, K') with (K_1, K_2) in $\mathcal{T}_{ct}$ to obtain the graph T. We prove that $T \in \Upsilon_{ct}$.

Since the graph T is connected, and has the same number of edges as $\mathcal{T}_{ct}$, the graph T is a tree. If the unique path P between 2 maximal cliques in T does not contain the edge $e = (K_1, K_2)$, the clique-intersection property holds (since the path P will be common between T and $\mathcal{T}_{ct}$). Now, consider 2 maximal cliques (K_3, K_4) such that the path P (in T) between K_3 and K_4 contains the edge e. We know that $K \cap K' = K_1 \cap K_2$, and, since both K, K' contain the set $K_3 \cap K_4$ (by definition of the clique-intersection tree), both K_1, K_2 contain the set $K_3 \cap K_4$ as well. Therefore, all maximal cliques on the path from K_1 to K_2 (in $\mathcal{T}_{ct}$) contain $K_3 \cap K_4$. Add the edge (K_1, K_2) in $\mathcal{T}_{ct}$ to obtain a cycle $[K_3, \dots K, K', \dots K_4, \dots K_2, K_1, \dots K_3]$ to obtain a cycle (this cycle contains the edge (K, K')). Now, every maximal clique on this cycle contains the set $K_3 \cap K_4$. The path P (of T) is obtained by removing the edge (K, K'), proving that the path P contains the intersection $K_3 \cap K_4$.

We have, therefore, verified that $T \in \Upsilon_{ct}$. Furthermore, observe that T has another edge in common with $\mathcal{T}_{mst}$, contradicting our assumption that $\mathcal{T}_{ct}$ is the clique-intersection tree with the highest number of common edges with $\mathcal{T}_{mst}$. Hence, we obtain $\Upsilon_{mst}^G \subseteq \Upsilon_{ct}^G$ (3.5). From 3.4 and 3.5, we obtain $\Upsilon_{mst}^G = \Upsilon_{ct}^G$. □

Given a chordal graph, we can construct its weighted clique intersection graph by finding all the maximal cliques (which can be accomplished efficiently using Algorithm 2). We now illustrate Kruskal's Algorithm ([13], [9, Chapter 4]) to find the Maximum Spanning Tree of any given tree, using which, one can find the MST of the weighted clique-intersection graph.

Given a connected graph $G = (V, E, w)$ with $w : E \to \mathbb{R}$ and $|V| = n$, we define the following:



(a) $\mathcal{T}_{ct}^{G}$ (b) T

Figure 3.3: Paths $K_3 \to K_4$ in trees T and $\mathcal{T}_{ct}^{G}$

- $T = (V, E_T, w|_{E_T})$ is our resulting spanning tree of maximum weight.
- S is the set of edges E, sorted in decreasing order of their weights.
- $s[0]$ is the first element of S.

Algorithm 4 Kruskal's algorithm

```
1: procedure KRUSKAL((G = (V, E, w))                    ▷ Finding a MST of G
2:     e ← s[0]
3:     while |E_T| ≠ n − 1 do                  ▷ We have the MST if n − 1 edges exist in T
4:         if E_T ∪ {e} has no cycle then
5:             T ← T ∪ {e}
6:         end if
7:         S ← S \ {e}
8:     end while
9:     return T = (V, E_T, w|_{E_T})                                  ▷ The MST is T
10: end procedure
```

Using the algorithm elucidated above, one can find the MST $\mathcal{T}_G$ of a weighted clique-intersection graph W_G, where G is chordal. We have also seen that the MST of W_G obeys the clique-intersection property. Observe that any clique-intersection tree of G is also a valid tree decomposition (due to 2.3.3) of G. Additionally, since all the vertices of the clique-

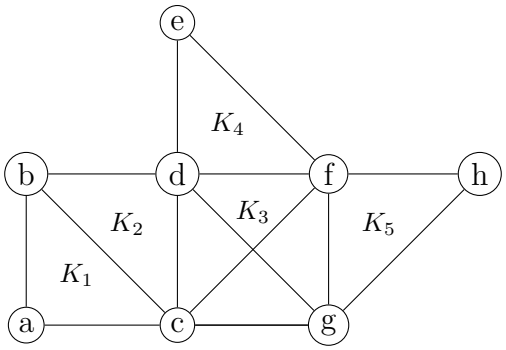
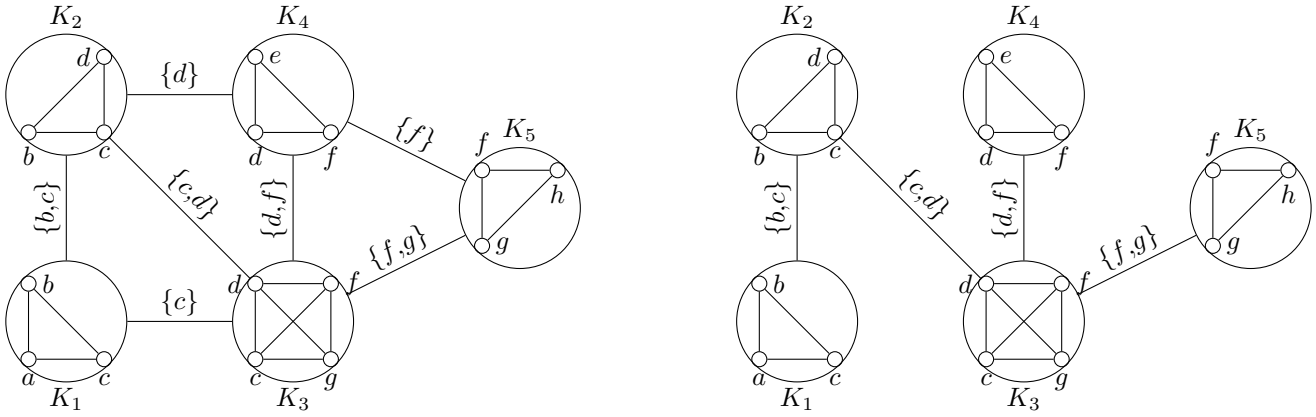


Figure 3.4: A chordal graph G

intersection tree are maximal cliques of G, the treewidth of the tree decomposition is exactly $\omega(G) - 1$.



(a) W_G (b) $\mathcal{T}_{ct}^G$

Figure 3.5: The weighted clique-intersection graph W_G and the corresponding MST $\mathcal{T}_{ct}^G$ of W_G.

Remark 3.1.1. *Every clique-intersection tree of a chordal graph G is a valid tree decomposition of G.*

Combining the above with Theorem 3.1.17, we have the following result.

Theorem 3.1.18. *For any chordal graph G, we have*

$$\Upsilon_{ct}^G = \Upsilon_{mst}^G = \Upsilon_{td}^G$$

Furthermore, $tw(G) = \omega(G) - 1$.

So, for any chordal graph G, we can quickly compute the tree decomposition.

3.1.5 Intersection Models

An *intersection graph* $G = (V, E)$ is formed in the following manner: we consider sets $S_1, S_2, \ldots S_n$, and vertices $v_1, v_2, \ldots v_n$ corresponding to these sets such that

$$E = \{(v_i, v_j) | S_i \cap S_j \neq \varnothing\}$$

The sets $S_1, S_2, \ldots S_n$ are called the *intersection model* of G. Intersection graphs form an interesting class of graphs, as any graph can be represented as an intersection graph (we omit the details of this claim, as it is not relevant to the focus of this thesis). For instance, chordal graphs are the intersection graphs of the induced subtrees of a tree ([12]).

One can place restrictions on the type of set to consider to obtain interesting subclasses of intersection graphs. One such graph class is the interval graph class. An *interval graph* is a graph whose intersection model consists of intervals on the real line. Interval graphs are a well-studied subclass of chordal graphs. Further information can be found in [1, Chapter 8].

3.2 Split Graphs

The following content, as well as supplementary details, can be found in [1, Chapter 6].

Definition 3.2.1. *Any graph $G = (V, E)$ whose vertex set V can be partitioned into a clique C and an independent set I is termed a* ***split graph****.*

We also define a subclass of split graphs, called complete split graphs, as follows.

Definition 3.2.2. *A* ***complete split graph*** *G is a split graph such that every vertex of the independent set is adjacent to every vertex of the clique.*

We will now look at some basic properties of split graphs, as well as give characterisations of the same.

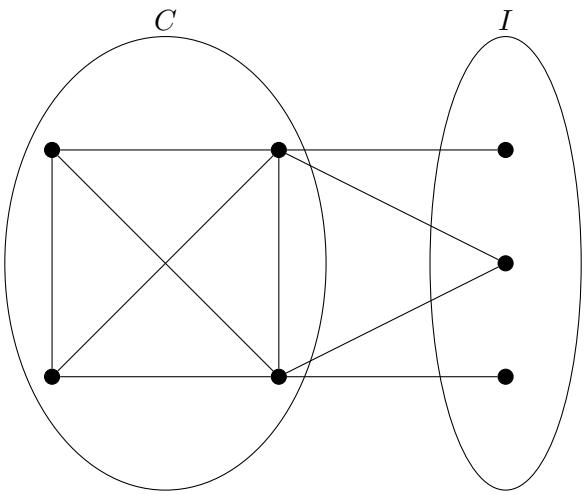


Figure 3.6: An example of a split graph with partitions C and I

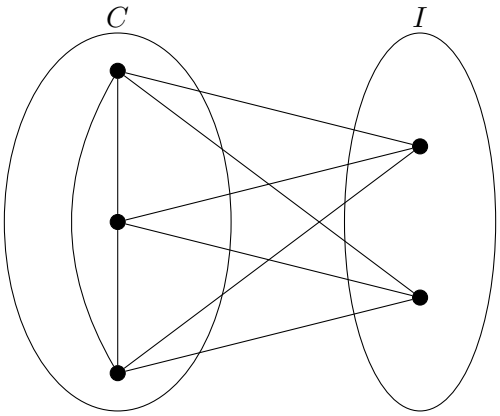


Figure 3.7: A complete split graph with partition C and I. Each vertex of I is adjacent to all vertices of C.

3.2.1 Basic Properties

The fact that cliques and independent sets are complements of one another leads directly to the theorem that follows.

Theorem 3.2.1. *[1, Chapter 6] The split graph class is closed under complements.*

The partition of V into C and I need not be unique. The theorem below enumerates the permissible conditions.

Theorem 3.2.2. *[1, Chapter 6] Let $G = (C \cup I, E)$ be a split graph, with C being the clique and I being the independent set. Then, either:*

1. *$|C| = \omega(G)$ and $|I| = \alpha(G)$, and the partition is unique.*

2. *$|C| = \omega(G) - 1$ and $|I| = \alpha(G)$, and there exists $i \in I$ such that $C \cup \{i\}$ is complete.*

3. *$|C| = \omega(G)$ and $|I| = \alpha(G) - 1$, and $\exists\ c \in C$ such that $I \cup \{c\}$ is independent.*

Proof. Observe that the clique and independent set can have at most 1 vertex in common. Hence, for any split graph G, $\omega(G) + \alpha(G) = |V|$ or $|V| + 1$.
If $\omega(G) + \alpha(G) = |V|$, then assume there is another partition of V into a clique C' and an independent set I'. Since $C \neq C'$, there exists atleast 1 vertex $u \notin C$ such that $\{u\} = C' \cap I$. We see that u is the unique vertex in C' that is not in C (if this were not the case, I would not be an independent set as it would contain 2 adjacent vertices). This means that $C \cup \{u\}$ is a clique in G, which has more than $\omega(G)$ vertices, contradicting the fact that $\omega(G)$ is the size of the maximum clique of G. Hence, if $\omega(G) + \alpha(G) = |V|$, then the partition $C \cup I$ is unique, and G is of the form (1).
Now, if $\omega(G) + \alpha(G) = |V| + 1$, $C \cup I \neq \varnothing$, then one of Case 2 or Case 3 is possible. If $|C| = \omega(G) - 1$ and $|I| = \alpha(G)$, $\exists\ i \in I$ such that $C \cup \{i\}$ is the maximum clique in G. Hence, $C \cup \{i\}$ is complete, and G is of the form (2). Similarly, if $|C| = \omega(G)$ and $|I| = \alpha(G) - 1$, $\exists\ c \in C$ such that $I \cup \{c\}$ is an independent set. Hence, $I \cup \{c\}$ is independent, and G is of the form (3). □

Split graphs are subclasses of chordal graphs with some additional structure imposed on them. The below theorem formalises and proves the same, and also provides a forbidden subgraph characterisation of split graphs.

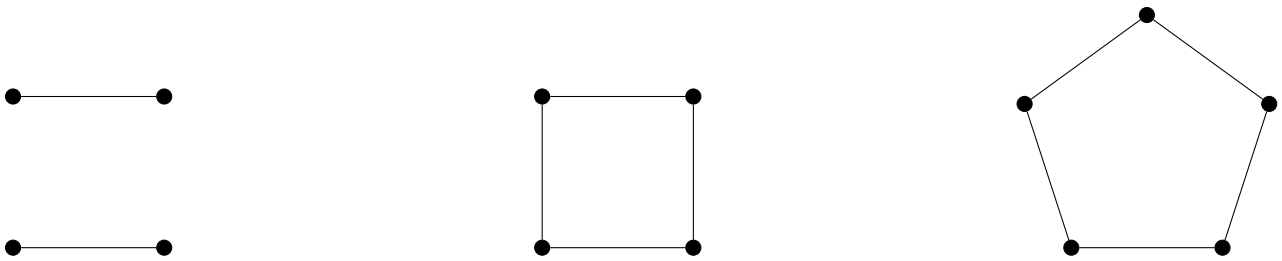


Figure 3.8: Forbidden subgraphs of split graphs

Theorem 3.2.3. *[1, Chapter 6] The following 3 statements equivalently define split graphs.*

1. *G is split.*
2. *G and $\overline{G}$ are both chordal.*
3. *G is $\mathcal{C}$-free for $\mathcal{C} = \{2K_2, C_4, C_5\}$.*

Proof. ((1) $\implies$ (2)) Here, we only need to prove that the split graph G is chordal, as we have already established that the complement of any split graph is split. See Theorem 3.2.1. Since perfect elimination orderings characterise chordal graphs by Theorem 3.1.4, we will now construct a perfect elimination ordering σ_G of the split graph G. Note that the vertex set $N[i]$ for each $i \in I$ is a subset of the clique C, and thus, $N[i]$ is complete for all $i \in I$. This implies that, $\forall\ i \in I$, i is a simplicial vertex of G. Thus, the first $|I|$ elements of σ_G are simply the vertices of I (can be placed in any order). Now, the graph $G[V \setminus I] = G[C]$ is a complete graph, implying that every vertex of $G[C]$ is simplicial in $G[C]$. Hence, to complete the ordering σ_G, we append all vertices of C to σ_G (in any order). This gives a perfect elimination ordering σ_G, proving that split graphs are chordal.
((2) $\implies$ (3)) Given G and $\overline{G}$ are both chordal. Observe that C_5 is self-complementary, and since G and $\overline{G}$ are both chordal, neither G or $\overline{G}$ contain an induced C_5. Additionally, note that $2K_2$ is the complement of C_4. Since G is chordal, it contains no induced C_4, and, thus, $\overline{G}$ contains no induced $2K_2$. The same argument can be used to prove that G contains no induced $2K_2$, by interchanging G and $\overline{G}$ in the argument above.
((3) $\implies$ (1)) Let C be the maximum clique of G so chosen such that $G[V - K]$ has the fewest possible edges. We will prove that $G[I]$ (where $I = V - K$) is an independent set.
Let us say $G[I]$ has an edge (u, v). Observe that neither u nor v is adjacent to every vertex

of C, owing to the maximality of C. Furthermore, if u and v have a common non-neighbour w in C, then $C \cup \{u, v\} \setminus \{w\}$ is a clique larger than C, contradicting the fact that C is a maximum clique. Hence, there exist unique vertices x and y such that $(u, x), (v, y) \notin E$. Since G contains no induced $2K_2$ or C_4, exactly one of (x, v) and (y, u) is an edge in G. Without loss of generality, say (x, v) is an edge in G. For any $z \in C \setminus \{x, y\}$, if neither (z, u) nor (z, v) are edges in G, then $G[\{z, y, u, v\}]$ induce a $2K_2$, which is not possible. However, if any one of $(z, u) \in E$ and $(z, v) \notin E$, then $G[\{z, x, u, v\}]$ induces a C_4, a contradiction. Thus, v is adjacent every vertex of $C \setminus \{y\}$, and $C' = C \cup \{v\} \setminus \{y\}$ is a maximal clique of same size as C.

Per our assumption, $G[V - C]$ has fewer edges than $G[V - C']$. Hence, $\exists\ w \neq v$ such that $(w, y) \in E$, but $(w, v) \notin E$. Now, if $(w, u) \notin E$, then $G[\{w, y, u, v\}]$ induces a $2K_2$, which is not permissible. Thus, $(w, u) \in E$. Additionally, if $(w, x) \in E$, then $G[\{w, u, v, x\}]$ induces a C_4, which is, again, not permissible. Now, observe that $G[\{w, u, v, x, y\}]$ induces a C_5, contradicting our assumption. Thus, $I = V - C$ is an independent set, proving that G is a split graph. □

3.3 Well-partitioned Chordal Graphs

We now study a new class of graphs called well-partitioned chordal graphs. This graph class was first introduced in [4] as a generalisation of split graphs and they are a subclass of chordal graphs. They can be used to close the complexity gap for problems that are intractable for chordal graphs but simple for split graphs. The next observation is crucial in arriving at the definition of well-partitioned chordal graphs. In any split graph, we can partition the vertex set into a clique C and an independent set I. Thus, if we picture C as the "central clique" surrounded by cliques of size 1 which are only adjacent to the vertices of C. we arrive at a structure that resembles a star graph (figure 3.9). We relax this condition in the following ways:

1. by allowing the partition's components to be set up in any kind of tree (instead of just a star), and

2. by allowing each partition to have an arbitrary number of vertices (unlike split graphs, in which, bar the central clique, every clique is of size 1).

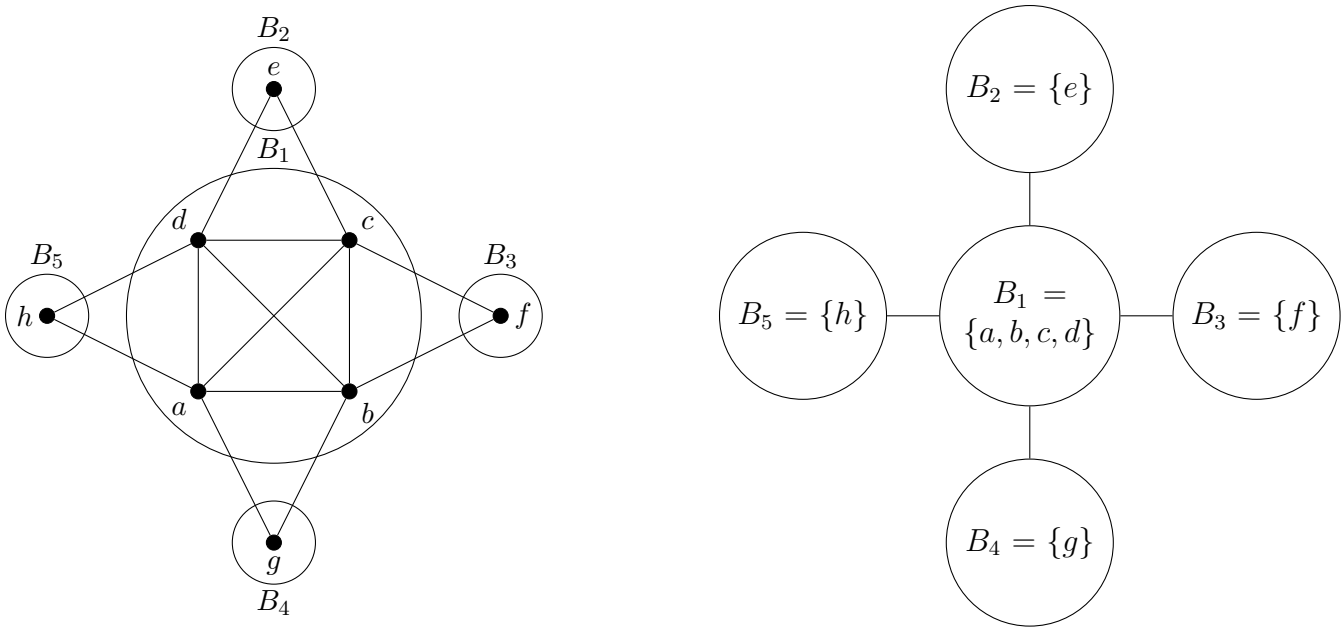


Figure 3.9: A split graph, with its partitions forming a star

Formally, we define well-partitioned chordal graphs as follows:

Definition 3.3.1. *[4] A **well-partitioned chordal graph** is a connected graph G with a partition $\mathcal{P}$ of $V(G)$ and an associated tree $\mathcal{T} = (\mathcal{P}, E(\mathcal{T}))$ such that:*

1. *$\forall X \in \mathcal{P}, G[X]$ is a clique in G.*

2. *For each pair of distinct $X, Y \in \mathcal{P}$ with $(X, Y) \notin E(\mathcal{T})$, $E(G[X, Y]) = \varnothing$.*

3. *For each edge $(X, Y) \in E(\mathcal{T})$, there exist $X' \subseteq X$ and $Y' \subseteq Y$ such that $E(G[X, Y]) = \{(x, y) | x \in X', y \in Y'\}$, i.e., $G[X' \cup Y']$ is a clique.*

The vertices of the tree $\mathcal{T}$ are referred to as "*bags*" and the tree is known as the *partition tree* of G.

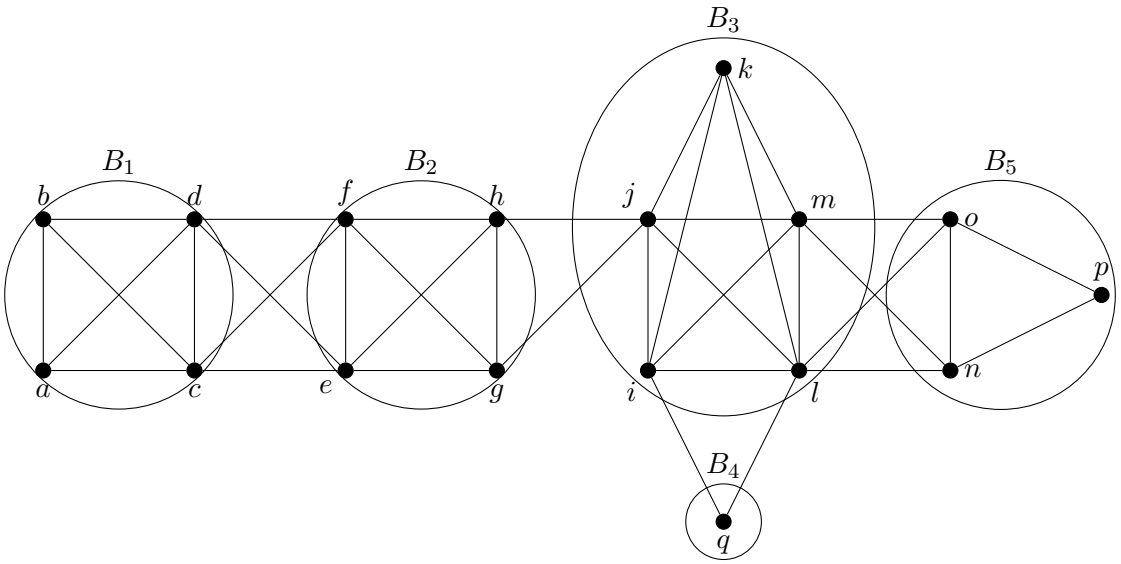


Figure 3.10: A well-partitioned chordal graph G

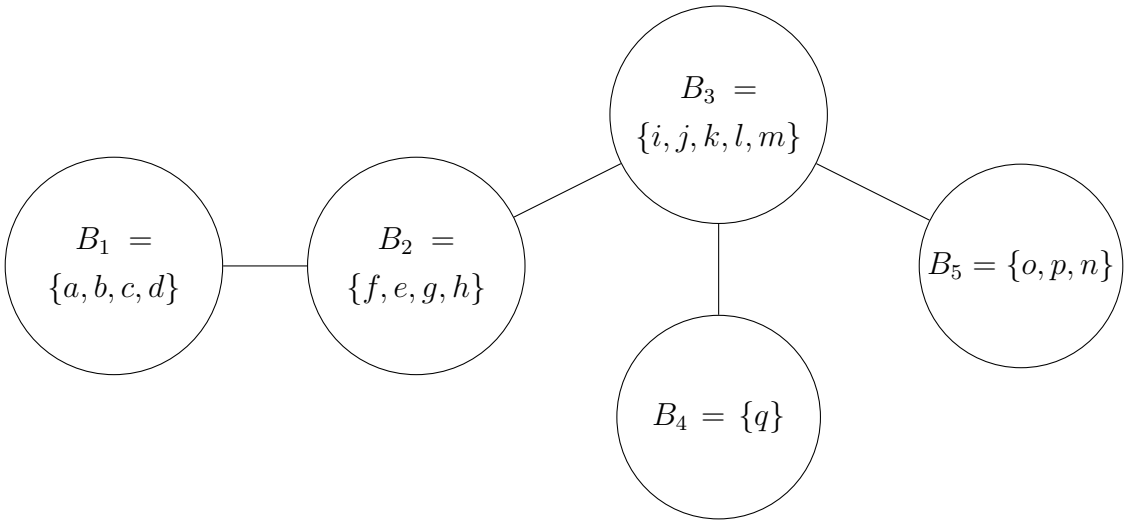


Figure 3.11: Partition tree $\mathcal{T}$ of the graph G

3.3.1 Basic Properties

Here, we mention 2 major results concerning well-partitioned chordal graphs. We first prove that they are, in fact, chordal. We then characterise these graphs in terms of their forbidden subgraphs.

Theorem 3.3.1. *Every well-partitioned chordal graph $G = (V, E)$ with partition tree $\mathcal{T} = (\mathcal{P}, E(\mathcal{T}))$ is chordal.*

Proof. Given a well-partitioned chordal graph G, we will give an explicit construction of a perfect elimination ordering σ_G of G. Observe that every leaf bag contains at least 1 simplicial vertex. This is because each bag is a clique, and, for any leaf bag L, either there exists at least one vertex v such that $N[v] \subseteq L$ (if this were not the case, then every vertex of the leaf bag will have a neighbor outside the bag, contradicting the fact that the bag would is a leaf bag of $\mathcal{T}$), or this bag is a subclique of some maximal clique. Now, let the set of simplicial vertices of G be S_G. Thus, the first $|S_G|$ elements of σ_G are simply the elements of S_G. Once we have exhausted all simplicial vertices of G, we generate a new well-partitioned chordal graph G' with vertex set $V \setminus S_G$ and partition tree $\mathcal{T}'$. We can continue the same process explained above till all the vertices are exhausted, thus giving us the perfect elimination ordering, proving that well-partitioned chordal graphs are also chordal. □

Theorem 3.3.2. *[4] All well-partitioned chordal graphs are $\mathbb{O}$-free, where $\mathbb{O}$ is shown as below. Furthermore, we can check if a graph is a well-partitioned chordal graph in polynomial time. If the graph is well-partitioned chordal, the algorithm additionally outputs the partition tree of the same.*

In lieu of providing the proof of this theorem here, we direct the reader to [4] for more information.

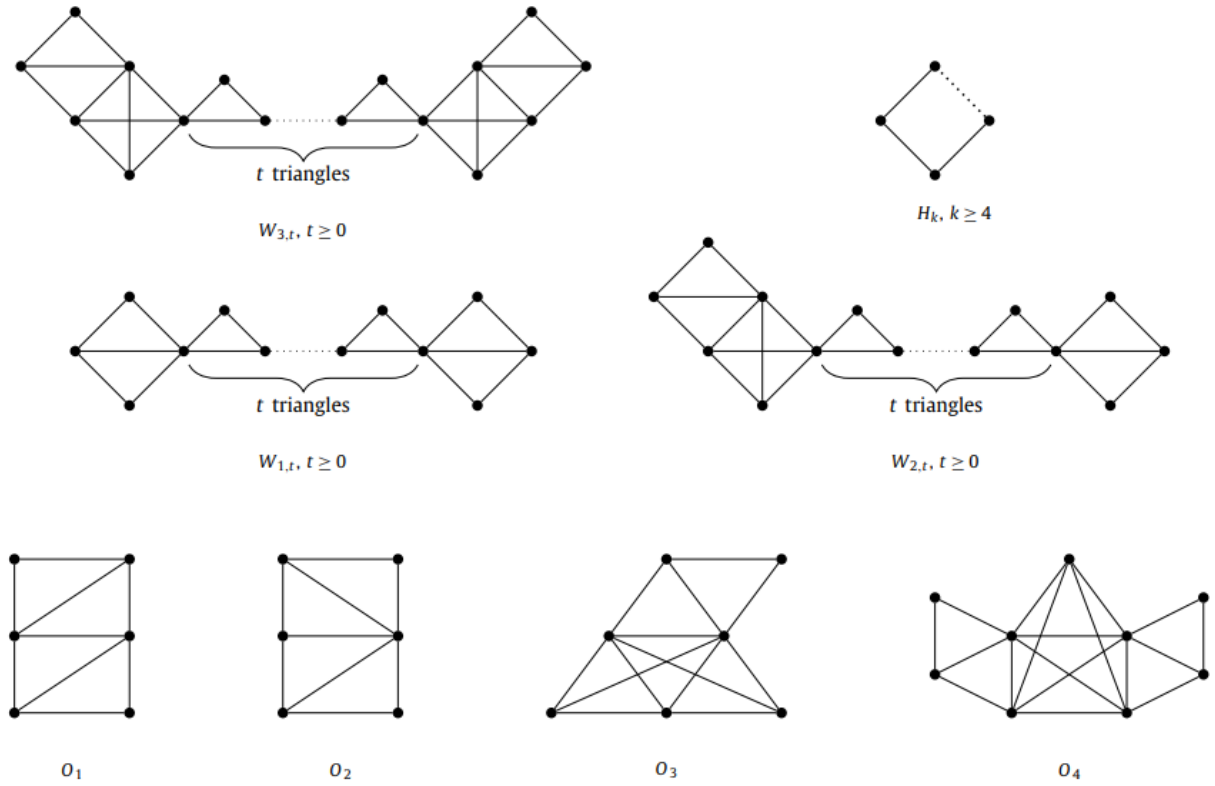


Figure 3.12: The set $\mathbb{O}$ of forbidden subgraphs of well-partitioned chordal graphs (H_k refers to cycle of length k) [Image credits: [4]]

Chapter 4

Vertex Deletion Problems

4.1 Introduction

In this chapter, we will give an overview on vertex deletion problems, which are a broad class of problems on graphs. Generally, given a graph $G = (V, E)$, a vertex deletion problem asks for a subset $S \subseteq V$ such that $G[V - S]$ satisfies some desirable property. For instance, an independent set problem is nothing but the vertex deletion problem to an edge-free graph (that is, the problem asks for a subset $S \subseteq V$ such that $G[V - S]$ is edge-free). Similarly, a maximum clique problem can be interpreted as the vertex deletion problem to a complete graph. We now study some basic concepts about vertex deletion problems, and illustrate some algorithms for the same.

Definition 4.1.1. *Given an input graph $G = (V, E)$, a* Π***-Vertex deletion problem*** *involves finding a minimum-sized set $V' \subseteq V$ such that the induced subgraph $G[V - S]$ satisfies the property* Π*.*

For properties Π satisfying a certain condition, Lewis and Yannakakis (in [14]) provided a dichotomy result which resolved the complexity classes of the Π-vertex deletion problems. To state the mentioned theorem, we require the following definition. If a graph property holds true for an infinite number of graphs but holds untrue for an infinite number of graphs, it is said to be *nontrivial*.

If the graph property Π is not non-trivial, then the Π-vertex deletion problem can be

solved in polynomial time. To see why this is the case, assume that the property Π is true only for finitely many graphs $G_1, G_2, \ldots G_n$. Then, we can simply find the largest G_i contained in our input graph, which can be done in polynomial time.

We now state the result.

Theorem 4.1.1. *[14] The* Π*-vertex deletion problem for non-trivial* Π *that are hereditary on induced subgraphs is NP-Complete.*

Subsequently, efforts in solving vertex deletion problems focused either on designing approximation and parameterized algorithms, or studying these problems by placing restrictions on the input graphs. For instance, we know that the Max Clique problem is NP-Complete on general graphs. However, if our input graph is restricted to the class of chordal graphs, we have seen that an efficient algorithm exists which provides the size of the maximum clique of the graph (refer to Algorithm 2). We focus on the latter approach, namely, we specify an input satisfying certain properties, and attempt to build efficient algorithms for solving vertex deletion problems on the same.

4.2 Some Basics

To study these problems, we need some rudimentary definitions and notations, which we specify below.

For a Π-vertex deletion problem,

- $\mathcal{C}$ denotes some unspecified hereditary graph class. To specify the same, we use block letters (for instance, we denote the class of chordal graphs as CHORDAL).
- For a graph G, if G belongs to a graph class $\mathcal{C}$, we write $G \in \mathcal{C}$.
- Unless specified, property Π is some specified hereditary graph class $\mathcal{C}$. A graph satisfies the property Π if it belongs to the specified graph class $\mathcal{C}$.
- The Π-vertex deletion problem can, thus, be rephrased as vertex deletion problem to some graph class $\mathcal{C}$.

- If our input graph is restricted to some graph class $\mathcal{C}_1$, and the vertex deletion problem is to some graph class $\mathcal{C}_2$, we denote this problem as $\mathcal{C}_1 \to \mathcal{C}_2$.
- $\mathcal{C}_1$ is termed as the input graph class, and $\mathcal{C}_2$ is termed as the target graph class.
- Thus, by denoting a vertex deletion problem as $\mathcal{C}_1 \to \mathcal{C}_2$, we convey that we are given a graph $G = (V, E) \in \mathcal{C}_1$, and we require the smallest subset $S \subseteq V$ such that $G[V - S] \in \mathcal{C}_2$.

For example, the Chordal $\to$ Split problem refers to the vertex deletion problem of transforming a chordal graph to a split graph, by deleting the fewest number of vertices.

We will now see some examples of vertex deletion problems. We are primarily concerned in problems where both our input and target graph classes are either chordal graphs, or their subclasses.

4.3 Vertex Deletion Problems

The following observation helps simplify the input classes to consider.

Remark 4.3.1. *[5] Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be 2 graph classes.*

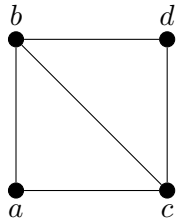
1. *If the problem $\mathcal{C}_1 \to \mathcal{C}_2$ has a polynomial time algorithm, then so does $\mathcal{C} \to \mathcal{C}_2$, for every subclass $\mathcal{C}$ of $\mathcal{C}_1$.*
2. *If the problem $\mathcal{C}_1 \to \mathcal{C}_2$ is NP-Complete, then, for any superclass $\mathcal{C}$ of $\mathcal{C}_1$, $\mathcal{C} \to \mathcal{C}_2$ is NP-Complete.*

Therefore, if we prove that a problem is NP-Complete for a subclass of chordal graphs, then we know that the problem is NP-Complete for chordal graphs as well. Similarly, finding an efficient algorithm on a larger class of graphs automatically solves the same problem on any subclass of this graph class.

We will now discuss algorithms for some vertex deletion problems. Our focus here is on designing simple polynomial-time algorithms based on structural properties.

Graph class	Forbidden subgraphs
Chordal	$C_n (n \geq 3)$
Split	$2K_2, C_4, C_5$
Complete Split	$\overline{P_3}, C_4$
Cluster	P_3
Block	$C_n, (n \geq 3)$, diamond

Forbidden subgraphs of some graph classes



The diamond graph

Figure 4.1: Forbidden subgraph characterisations (sourced from [15])

4.3.1 The Well-Partitioned Chordal → Split Problem

Theorem 4.3.1. *For a given well-partitioned graph $G = (V, E)$ with partition tree $\mathcal{T} = (\mathcal{P}, E(\mathcal{T}))$, let $H = (C \cup I, E')$ be the split subgraph of G of maximum size. Then $C \in \mathcal{P}$.*

Proof. For a split subgraph H of the well-partitioned chordal graph G, C cannot be a union of cliques (as C would violate the definition of a clique). Thus, either $C \subsetneq X$ or $C = X$ for some $X \in \mathcal{P}$. If $C \subsetneq X$, we can simply append all the remaining vertices of X to H. Observe that the independence of I is unaffected by performing this procedure. We have, thus, obtained a split subgraph of G of higher size than H, contradicting our assumption. Thus, for a maximum split subgraph H of G, we have $C = X$ for some $X \in \mathcal{P}$. □

Using this theorem, we arrive at the following algorithm. In the below algorithm, $IS[G]$ (for a graph G) denotes the maximum independent set of G.

Theorem 4.3.2. *Algorithm 5 solves the* Well-Partitioned Chordal → Split *problem in polynomial time.*

Proof. Let the input to Algorithm 5 be a well-partitioned chordal graph G of order n, and let $\mathcal{T}$ be the partition tree of the same. From Theorem 4.3.1, we know that any clique forming a split graph is a bag of the partition tree. Thus, for every bag $K \in \mathcal{P}$, we assume that K forms the clique of the split graph, and find the biggest independent set of $G[V - K]$, producing a split subgraph of G. Each such step takes polynomial time, and the size of

Algorithm 5 Well-Partitioned Chordal → Split

Input: $G = (V, E), \mathcal{T} = (\mathcal{P}, E(\mathcal{T}))$ ▷ Given a WPCG with partition tree

Output: VD ▷ VD is the required vertex deletion set

1: **procedure** SVD($(G = (V, E))$

2: $S \leftarrow \varnothing$

3: **for all** $X \in \mathcal{P}$ **do**

4: $C_X \leftarrow X$ ▷ C_X is our candidate clique of resultant split graph

5: $G_X \leftarrow G[V - V[X]]$

6: $I_X \leftarrow IS[G_X]$ ▷ Independent set of resultant split graph

7: $S_X \leftarrow G[C_X \cup I_X]$ ▷ Our resultant split graph with C_X as clique

8: $S \leftarrow S \cup S_X$

9: **end for**

10: **return** $VD = V - \max_{X \in \mathcal{P}} \Big\{ |S_X| \Big\}$

11: **end procedure**

the partition tree is at most n. Thus, we arrive at a polynomial time algorithm for the Well-Partitioned Chordal → Split problem. □

4.3.2 The Interval → Cluster Problem

The following algorithm was first given and proved in [5].

Given an interval graph $G = (V, E)$ with its interval model, we are required to transform it into a cluster graph by removing the fewest number of vertices from G. We denote the left and right endpoints of the interval of a vertex v by $l(v)$ and $r(v)$ respectively.
Let us say that H is the largest cluster subgraph of G, and that it has m cliques. For each clique $K_i, 1 \leq i \leq m$, we can find 2 endpoints such that

$$a_i = \min_{v \in K_i} l(v)$$

$$b_i = \max_{v \in K_i} r(v)$$

Now, observe that the intervals for all vertices $u \in K_i$ are contained in the interval $[a_i, b_i]$.

Additionally, each of the m such intervals is pairwise disjoint i.e. there are no edges between 2 distinct cliques in H. Hence, if we find m such pairs of $[a_i, b_i]$, our cliques will simply be the maximal cliques in the graph induced by the vertices corresponding to the intervals contained in $[a_i, b_i]$.
To solve this, we build the following weighted interval graph $\mathcal{W} = (V_{\mathcal{W}}, E_{\mathcal{W}})$ with weight function $w : V_{\mathcal{W}} \to \mathbb{R}$:

- For each $l(v_l)$ and each $r(v_r)$ with $l(v_l) < r(v_r)$ (and considering $v_l = v_r$), we construct a vertex with its associated interval equal to $[l(v_l), r(v_r)]$. Let each vertex be called $u(v_l, v_r)$.
- The weight $w(u(v_l, v_r))$ is nothing but the size of the maximum clique of the subgraph induced by $\{v : l(v_l) \leq l(v) < r(v) \leq r(v_r)\}$.
- As with any interval graph, 2 vertices are adjacent if and only if their intervals share at least 1 common element.

Since this is an interval graph, we can find the maximum cliques (i.e. the weights of the vertices of $\mathcal{W}$) in polynomial time. Since our target graph is a cluster graph, we require each clique to be disjoint from all the other cliques. To ensure the same, we find the maximum-weight independent set of $\mathcal{W}$. Thus, we have found a maximum-sized cluster graph given an input interval graph. We omit the proof of the algorithm, and refer the reader to [5] for the details.

Chapter 5

Some New Results

In this chapter, we provide some algorithms for vertex deletion problems on subclasses of chordal graphs, mainly on the well-partitioned chordal graph class. In these algorithms, we have the operation argmin which takes, as inputs, a collection of sets, and returns the set corresponding to the minimum cardinality (for instance, given 2 sets A, B with $|A| < |B|$, argmin$\{|A|, |B|\}$ returns the set A).

5.1 The Split $\rightarrow$ Cluster Problem

We elucidate a different polynomial time algorithm for this problem than the one given in [5]. Given any split graph $G = (V, E)$ with partition $C \cup I$, we want to find the smallest vertex set $S \subseteq V$ such that $G[V - S]$ is a cluster graph. Let the resultant cluster graph be $H = (V', E')$, with cliques $K_1, K_2, \ldots K_q$ for some $q \in \mathbb{N}$.
The following observation is crucial. For any $v \in I$, one of three cases are possible:

C1. $v \notin V'$

C2. $v \in V'$ with $d_H(v) = 0$

C3. $v \in V'$ with $d_H(v) \neq 0$

Here, $d_H(v)$ is the degree of v in the graph H. Thus, we can design a basic dynamic

programming algorithm to solve our problem as follows. Here, $I_{G'}$ denotes the independent set of the split graph G'.

Theorem 5.1.1. *Algorithm 6 solves the* Split → Cluster *problem in polynomial time.*

Proof. For case C1, we add v to the solution and solve the problem for the graph $G' - v$. On the other hand, if we have that C2, we add $N(v)$ to the solution, and solve the problem for the graph $G' - N[v]$. These cases are covered in line 5. The case C3 is covered in line 12. Since each subproblem can be solved in polynomial time, and the number of vertices in I_G is bounded by n, the algorithm takes polynomial time. □

Algorithm 6 Split → Cluster

Input: $G = (C \cup I, E)$ ▷ Given a split graph with the partition

Output: $S[G]$ ▷ $S[G]$ is the required vertex deletion set

1: **procedure** CVD($(G = (C \cup I, E))$
2: $dp[\varnothing] \leftarrow \varnothing$
3: $G' \leftarrow G$
4: **while** G' not a cluster graph **do** pick $v \in I_{G'}$
5: $dp[G'] \leftarrow \text{argmin}\{|\{v\} \cup dp[G' - v]|, |N(v) \cup (dp[G' - N[v]|\}$ ▷ Cases C1 and C2
6: $G' \leftarrow G' \setminus dp[G']$
7: **if** G' is a cluster graph **then**
8: $dp[G'] \leftarrow \varnothing$
9: **end if**
10: **end while**
11: **for all** $v \in I$ **do**
12: $S_v = N[v] \cup \alpha(G - N[v])$ ▷ Case C3
13: **end for**
14: $S = \underset{v \in I}{\text{argmin}}\{|S_v|\}$
15: $S[G] = \text{argmin}\{|dp[G]|, |G \setminus \omega(G)|, |G \setminus \alpha(G)|, |G \setminus S|\}$
16: **return** $S[G]$
17: **end procedure**

Algorithm 7 Split → Complete Split

Input: $G = (C \cup I, E)$ ▷ Given a split graph with the partition

Output: VD ▷ VD is the required vertex deletion set

1: **procedure** CSVD($(G = (C \cup I, E))$

2: $VD \leftarrow SVD(\overline{G})$ ▷ The procedure SVD refers to Algorithm 6

3: **return** VD

4: **end procedure**

Given a split graph G, Algorithm 6 helps transform it into a P_3-free graph (cluster graphs are P_3-free, refer to Section 4.1). Observe that complete split graphs are split graphs that are $\overline{P_3}$-free. This fact, combined with the result that split graphs are closed under the complement operation (Theorem 3.2.1) gives us the following result (the same idea was used in [5] to solve the Split → Complete Split problem).

Corollary 5.1.2. *Algorithm 6 solves the* Split → Complete Split *problem.*

More explicitly, given a split graph $G = (V, E)$, we execute Algorithm 6 on the complement graph $\overline{G}$ to obtain a set S such that $\overline{G}[V - S]$ is P_3-free. Thus, removing the set S from G results in the graph G being $\overline{P_3}$-free, making the resultant graph a complete split graph (as split graphs are already C_4-free). We state the same in Algorithm 7 for brevity.

5.2 The Well-Partitioned Chordal → Complete Split Problem

We claim that the Well-Partitioned Chordal → Complete Split vertex deletion problem can be solved by employing the Split → Complete Split algorithm elucidated above (Algorithm 7). We first provide the pseudocode for the algorithm, and then argue for its correctness.

We define the following notations which are necessary for the algorithm.

- $G = (V, E)$ is the given well-partitioned chordal graph, and $\mathcal{T} = (\mathcal{P}, E_{\mathcal{T}})$ is the parti-

tion tree of G. For any $B \in \mathcal{T}, V[B]$ refers to the set of vertices in clique B.,

- $H = (C \cup I, E')$ is the largest complete split subgraph of G, where C and I are the clique and independent set, respectively.
- For any $B \in \mathcal{T}, N_{\mathcal{T}}(B)$ is the set of neighbors of B in $\mathcal{T}$.
- Additionally, for $B, B' \in \mathcal{T}, NE(B, B') = \{v \in V[B'] | N(v) \cap B \neq \varnothing\}$.
- The output of implementing Algorithm 7 with input graph R is denoted by $A[R]$
- For a clique $B \in \mathcal{P}$, we contract all the vertices $V[B]$ to obtain a single vertex b. We denote this operation by $C[B]$.

Algorithm 8 Well-Partitioned Chordal → Complete Split

Input: $G = (V, E), \mathcal{T} = (\mathcal{P}, E(\mathcal{T}))$ ▷ Given a WPCG with partition tree

Output: VD ▷ VD is the required vertex deletion set

procedure CSVD($(G = (V, E))$
  $S \leftarrow \varnothing$
  for all $X \in \mathcal{P}$ **do**
    for all $Y \in N_{\mathcal{T}}(X)$ **do**
      $I \leftarrow \{y \mid y = C[Y]\}$
      $G_X \leftarrow G[X \cup I]$
      $S_X = A[G_X]$
    end for
  end for
  $S_G \leftarrow \underset{X \in \mathcal{P}}{\text{argmin}} \Big\{ |S_X| \Big\}$
  return $VD = \text{argmin} \Big\{ |S_G|, |\omega(G)|, |\alpha(G)| \Big\}$
end procedure

Theorem 5.2.1. *Algorithm 8 solves the* Well-Partitioned Chordal → Complete Split *problem in polynomial time.*

Proof. Let $H = (V_H, E_H)$ with $V_H = C \cup I$ be the largest complete split subgraph in a well-partitioned chordal graph G of order n, and let $\mathcal{T}$ be the partition tree of the same.

Let $C \subset K$ for some $K \in \mathcal{T}$. Then, $|I \cap K'| \leq 1$ for all $K' \in N_{\mathcal{T}}(K)$. If this were not the case, then we would have 2 vertices adjacent to each other in I, contradicting the definition of a complete split graph. Further, V_H does not contain any vertices from any bag $B \notin \{K\} \cup N_{\mathcal{T}}(K)$, as no such vertex will be adjacent to the vertices of C. Now, let $NE(K', K) = \{v \in K' | N(v) \cap K \neq \phi\}$, where $K' \in N_{\mathcal{T}}(K)$. Observe that, for any $v_1, v_2 \in NE(K', K), N(v_1) \cap B = N(v_2) \cap B$, by definition of well-partitioned chordal graphs. This is to say that, any vertex in K' adjacent to a vertex in K will be adjacent to the same set of vertices in K, as any other vertex in K' with neighbors in K. Thus, every clique in $N_{\mathcal{T}}(K)$ contributes at most one vertex to the complete split subgraph. For every $K' \in N_{\mathcal{T}}(K)$, contract the vertices of K' into a single vertex k', and remove all other bags of $\mathcal{T}$ to obtain a split graph. For every such split graph, we execute Algorithm 7 to obtain a complete split subgraph H_K. For every $k' \in V(H_K)$, we select a vertex $v_{K'}$ in K' such that $v_{K'}$ is adjacent to a vertex in K, thus obtaining a complete split subgraph of G.
Each such instance takes polynomial time, and since the size of the partition tree is bounded by n, the process takes polynomial time. □

5.3 The Well-Partitioned Chordal → Cluster Problem

Some notation required:

- $\mathcal{L}$ is the set of leaves of the partition tree $\mathcal{T}$.
- $N(L)$ refers to the neighborhood set of the vertices of L.
- $B(L)$ refers to the set of vertices of L that have neighbors outside the set L.
- L' is the neighbor of L in $\mathcal{T}$.

Let us consider a well-partitioned chordal graph $G = (V, E)$ with the partition tree $\mathcal{T} = (\mathcal{P}, E_{\mathcal{T}})$. Let $H = (V', E')$ be the largest cluster subgraph of G, For any leaf bag $L \in \mathcal{P}$, the following 3 cases are possible:

1. $L - B(L)$ forms a disjoint clique of H (if $L = B(L)$), this is equivalent to removing L from G).

2. L forms a disjoint clique of H.

3. $B(L, L')$ forms a disjoint clique of H.

For 1, we need to remove $B(L)$ from the graph to make $L - B(L)$ a disjoint clique. For 2, observe that we need to remove $B(L')$ from the graph to make L a disjoint clique. For 3, we need to remove all vertices from $L \cup L'$ that are not in $B(L, L')$. The sets S_1, S_2 correspond to the first 2 cases elucidated above. Observe that, for case 3, we need $B(L) = L$. Thus, based on whether $B(L) = L$ or $B(L) \subsetneq L$, we modify S_3 accordingly.

Theorem 5.3.1. *Algorithm 9 solves the* Well-Partitioned Chordal $\rightarrow$ Cluster *problem.*

Proof. Proof by induction on the size of the partition tree. Let $|\mathcal{T}| = l$. For $l = 1$, nothing to do as the graph is already a complete graph. Assume true for $k < l$. Now, if $|\mathcal{T}| = l$, we prove each of the sets appended to the solution set either generates a disjoint clique, or removes obstruction to cluster in the graph induced by $V = V(L) \cup V_{\mathcal{T}}(N(L))$. Let L be the new leaf added to $\mathcal{T}$. We go case by case.

Case 1: $B(L) \subsetneq L$

1. S_1: makes $L - B(L)$ a disjoint clique.

2. S_2: makes L a disjoint clique.

3. S_3: makes $B(L, L')$ a candidate for a disjoint clique (i.e., we modify the leaf L to $K = L \setminus B(L)$ so that $K = B(K)$, which implies we can have $B(K, L')$ as a disjoint clique.

In either of the cases, we see that the graph induced by $V = V(L) \cup V_{\mathcal{T}}(N(L))$ becomes P_3-free.

Case 2: $B(L) = L$

1. S_1: eliminates L from the graph.
2. S_2: makes L a disjoint clique.
3. S_3: makes $B(L, L')$ a disjoint clique.

In either of the cases, we see that the graph induced by $V = V(L) \cup V_{\mathcal{T}}(N(L))$ becomes P_3-free.

To prove our solution is optimal, we use induction on the size of the partition tree. Trivially true for $l = 1$, assume true for $k < l$ and $|\mathcal{T}| = l$ (observe that, the number of vertices of the partition tree is bounded by n, and thus the procedure takes polynomial time). For any bag L of the tree, our cases are (here, $B(L) \subsetneq L$) :

- For 1, we need to add the boundary of L to the solution set to make $L - B(L)$ disjoint, and solve the problem for $G' - L$.
- For 2, we need to add $B(L')$ to the solution set to make L disjoint.
- For 3, we need to add all vertices of the 2 bags except the shared boundary to the solution set, to make $B(L, L')$ disjoint.

Similar arguments hold if $B(L) = L$.

Now, assume $B(L) = L$. Our cases are:

- For 1, since $L = B(L)$ we need to add L itself to the solution set, and solve the problem for $G' - L$.
- For 2, we need to add $B(L')$ to the solution set to make L disjoint.
- For 3, we need to add all vertices of the 2 bags except the shared boundary to the solution set, to make $B(L, L')$ disjoint.

□

Algorithm 9 Well-Partitioned Chordal → Cluster

Input: $G = (V, E), \mathcal{T}$ ▷ Given a WPCG with the partition tree

Output: $Sol[G]$ ▷ $Sol[G]$ is the required vertex deletion set

procedure CVD($(G = (V, E))$
  $dp[\varnothing] \leftarrow \varnothing$
  while G' is not a cluster graph **do** pick $L \in \mathcal{L}$
    if $B(L) \subsetneq L$ **then**
      $S_1 \leftarrow B(L) \cup dp[G' - L]$
      $S_2 \leftarrow (N(B(L)) \cup dp[G' - L]$
      $S_3 \leftarrow (L' - B(L, L')) \cup dp[G' - (L \cup L'))]$
      $dp[G'] \leftarrow \text{argmin}\{|S_1|, |S_2|, |S_3|\}$
      $G' \leftarrow G' \setminus dp[G']$
    else if $B(L) = L$ **then**
      $S_1 \leftarrow L \cup dp[G' - L]$
      $S_2 \leftarrow N(L) \cup dp[G' - L]$
      $S_3 \leftarrow N(N(L)) \cup dp[G' - (L \cup L')]$
      $dp[G'] \leftarrow \text{argmin}\{|S_1|, |S_2|, |S_3|\}$
      $G' \leftarrow G' \setminus dp[G']$
    end if
    if G' is a cluster graph **then**
      $dp[G'] \leftarrow \varnothing$
    end if
  end while
  $Sol[G] = \text{argmin}\{|dp[G]|, |G \setminus \omega(G)|, |G \setminus \alpha(G)|\}$
  return $Sol[G]$
end procedure

Chapter 6

Conclusions and Open Problems

The thesis project contained a study on vertex deletion problems, primarily on subclasses of chordal graphs. We surveyed crucial properties of chordal graphs and their subclasses in Chapter 3. We also examined the newly introduced class of well-partitioned chordal graphs.

Post this, in Chapter 4, we studied the basic theory of vertex deletion problems. We looked at broad ideas for approaching vertex deletion problems and surveyed existing results on the same. We specifically focused on vertex deletion problems where we are given an input graph class and a target graph class, and we want to delete the fewest number of vertices to transform the input class to the target graph class.

In Chapter 5, we attacked the following 3 problems: the Split $\rightarrow$ Cluster problem, the Well-Partitioned Chordal $\rightarrow$ Complete Split problem and the Well-Partitioned Chordal $\rightarrow$ Cluster problem. We provided basic polynomial-time algorithms for the above problems, exploiting their structural properties. However, there is a vast array of still unresolved vertex deletion problems on these classes. In particular, it would be interesting to explore the computational complexity of other unresolved problems such as the Chordal $\rightarrow$ Cluster problem and the Chordal $\rightarrow$ Unit Interval problem.

Bibliography

[1] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.

[2] Pinar Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3):297–317, 2006.

[3] Frédéric Havet, Cláudia Linhares Sales, and Leonardo Sampaio. b-coloring of tight graphs. *Discrete Applied Mathematics*, 160(18):2709–2715, 2012.

[4] Jungho Ahn, Lars Jaffke, O-joung Kwon, and Paloma T Lima. Well-partitioned chordal graphs. *Discrete Mathematics*, 345(10):112985, 2022.

[5] Yixin Cao, Yuping Ke, Yota Otachi, and Jie You. Vertex deletion problems on chordal graphs. *Theoretical Computer Science*, 745:75–86, 2018.

[6] Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, August 2005.

[7] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.

[8] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[9] Jon Kleinberg and Éva Tardos. *Algorithm Design*. Addison Wesley, 2006.

[10] László Lovász. A characterization of perfect graphs. *Journal of Combinatorial Theory, Series B*, 13(2):95–98, 1972.

[11] Jean RS Blair and Barry Peyton. An introduction to chordal graphs and clique trees. In *Graph theory and sparse matrix computation*, pages 1–29. Springer, 1993.

[12] Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974.

[13] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[14] John M Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

[15] H. N. de Ridder et al. Information System on Graph Classes and their Inclusions (ISGCI). https://www.graphclasses.org.

[16] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[17] P Sreenivasa Kumar and CE Veni Madhavan. Clique tree generalization and new subclasses of chordal graphs. *Discrete Applied Mathematics*, 117(1-3):109–131, 2002.